

Pràctiques de Sistemes Digitals i Microprocessadors Curs 2023-2024

Pràctica 1 - Fase 3

LSBattleShip

| Alumnes | Login | Nom |
|---------|-----------------------|---------------------------------|
| | oriol.aparicio | Oriol Aparicio Casanovas |
| | hicham.naf | Hicham Naf |

| Entrega | Placa | Memòria | Nota |
|---------|-------|---------|------|
| | | | |

| | |
|------|-------------------|
| Data | 10/03/2024 |
|------|-------------------|

Index

| | |
|--|-----------|
| 1. Resum de l'enunciat | 2 |
| 2. Plantejament del software | 3 |
| 2.1 EUSART | 3 |
| 2.1.1 Lectura i Escriptura | 3 |
| 2.1.2 Mostrejat dels Menús | 4 |
| 2.1.3. Lectura Opcions | 6 |
| 2.1.4 Mostratge matriu | 7 |
| 2.1.5 Opcional 1.3: Menú Avançat Automàtic | 11 |
| 2.1.6 Extres | 13 |
| 2.2 EEPROM | 14 |
| 2.2.1 Distribució de la memòria | 14 |
| 2.2.2 Lectura i escriptura de l'EEPROM | 15 |
| 2.2.3 DEBUG EEPROM | 18 |
| 2.2.4 Opcional 2: Global View | 21 |
| 2.3 ADC | 23 |
| 2.3.1 Joystick | 23 |
| 2.3.2 Opcional 3: Brillantor de la pantalla variable | 25 |
| 2.4 Opcional 4: Rainbow | 26 |
| 2.5 Opcional 5: Nuke | 29 |
| 2.6 Opcional 6: El cançoner | 35 |
| 3. Configuracions del microcontrolador | 38 |
| CONFIGURACIONS EUSART | 38 |
| CONFIGURACIONS ADC | 40 |
| 4. Esquema elèctric | 40 |
| 5. Diagrama d'activitat del software | 41 |
| 6. Conclusions i problemes observats | 42 |
| 7. Planificació | 43 |

1. Resum de l'enunciat

Per a realitzar la pràctica de LSBattleShip implementarem digitalment el joc d'enfonsar vaixells BattleShip entre dues persones.

El funcionament consistirà a fer que, els dos usuaris introduceixin les coordenades dels seus 5 vaixells per torns. Un cop distribuïts comença el joc, llavors per torns els jugadors hauran d'atacar les caselles del taulell i el sistema comprovarà si hi havia un vaixell contrincant. Finalment, el joc finalitzarà quan un dels jugadors hagi encertat la posició de tots els vaixells contrincants.

En aquesta fase 3 expandirem el funcionament ja implementat a la fase 1 i 2, afegint l'ús d'un joystick per a moure el cursor, la possibilitat de comunicar la placa amb l'ordinador, mitjançant la EUSART, mostrant les accions que es vagin fent, a més a més de disposar de noves funcionalitats. Finalment, també podrem prendre una partida que s'hagi deixat a mitges, indicant-ho al menú inicial, en cas que s'hagi desconnectat la placa o s'hagi fet un reset involuntari.

Per a fer tota aquesta expansió de funcionalitats, ampliarem el codi implementat anteriorment a la fase 2, de manera que puguem afegir totes les noves especificacions.

2. Plantejament del software

En aquesta 3ra fase tenim 3 grans funcionalitats que hem d'implementar, la EUSART, la EEPROM i l'ADC. A continuació les anirem explicant en aquest mateix ordre.

2.1 EUSART

Abans de posar-nos a modificar el codi de fase 2, hem creat un nou projecte i hem començat amb la comunicació EUSART, enviar i rebre un caràcter.

Quan hem aconseguit això, hem creat un menú a la taula de la flash i hem fet les funcions per mostrar un menú i llegir una opció, mostrar una matriu posant valors aleatoris en la RAM per provar.

Finalment, ja hem començat a implementar les funcions en la fase 2.

2.1.1 Lectura i Escriptura

Per enviar un símbol pel canal TX, hem de posar el valor al registre TXREG i llavors ja s'encarrega d'enviar-ho pel canal sèrie.

Hem implementat la funció *MOSTRAR_CARACTER* que abans de cridar-la, guardem el valor del símbol en ASCII al registre **VALUE**, i quan entrem a la funció comprovem el bit TRMT del registre **TXSTA** per comprovar si el canal està disponible, si està encara ocupat ens esperem, quan estigui lliure carreguem el valor al **TXREG** i acabem.

```

MOSTRAR_CARACTER
    BTFSS   TXSTA, TRMT, 0
    GOTO    MOSTRAR_CARACTER
    MOVFF   VALUE, TXREG
    RETURN

```

Per llegir del RX, hem de consultar el bit RCIF del registre PIR1 que quan val 0 vol dir que no tenim cap dada nova disponible al registre RCREG i quan val 1 vol dir que sí que hi ha una dada nova, aquest registre actua com una pila que pot acumular fins a 4 dades, com que l'anem consultant molt sovint, no tenim problema que s'acumulin més de 4 dades.

Al contrari de l'escriptura, aquí no ens volem esperar indefinidament esperant al fet que ens arribi alguna dada pel canal RX, sinó que tenim moltes coses a fer com actualitzar la matriu, llegir els pulsadors, canal analògic...

Per aquest motiu, en el bucle principal anem consultant el bit RCIF, si val 0 perquè no hi ha cap dada tornem i fem altra cosa, si tenim una dada llavors sí que la llegim i la gestionem.

```
READ_OPTION_1
    BTFSS PIR1, RCIF, 0
    RETURN
    MOVFF RCREG, CONTINGUT
    ...
```

2.1.2 Mostrejat dels Menús

Enviar un menú sencer pel canal TX símbol per símbol seria una tasca molt tediosa i llarga si ho hem de fer amb una funció.

Per aquest motiu utilitzem la taula flash on allà guardem el text dels diferents menús i quan vulguem mostrar algun, anem a la regió de la taula flash que toca i amb un bucle anem llegint i enviant.

Per exemple si volem mostrar el menú principal, el tenim a la flash d'aquesta manera.

```
TAULAEUSART
:MENU PRINCIPAL
DB "\n\rSelect an option: \n\r", "1 - New Game \n\r"
DB "2 - Resume Game\n\r"
DB "> ", "?"
```

On en ara posem una etiqueta al començament de tot sense assignar-li cap valor perquè ja ho farà la PIC a on l'hi convingui.

Una manera d'escriure el menú en lloc d'anar escrivint valor per valor, és entre cometes posar el text que vols i ja es distribueix bé per la flash.

```
READ_FLASH
    CALL INIT_FLASH
BUCLE_ESCRIPTURA_MENU_1
    TBLRD*
    MOVFF TABLAT, VALUE
    MOVLW .63
    SUBWF VALUE, 0, 0
    BTFSC STATUS, Z, 0
    RETURN
    CALL MOSTRAR_CARACTER
    GOTO BUCLE_ESCRIPTURA_MENU_1
```

Amb aquesta funció, enviem tot el menú pel canal TX, primer cridem la funció *INIT_FLASH* que ens inicialitza els punters de la flash a la primera adreça del menú, llavors llegim el contingut i el guardem a **VALUE**, per saber quan hem acabat d'enviar un Menú, hem afegit un símbol al final que és un interrogant, si el símbol que volem enviar no és '?' llavors cridem la funció *MOSTRAR_CARACTER* per

enviar-lo i tornem a l'inici del bucle per llegir el següent caràcter, sí, en canvi, el símbol és '?', vol dir que hem acabat de mostrar tot el menú i fem return.

```
INIT_FLASH
    MOVLW  LOW(TAULAEUSART)
    MOVWF  TBLPTRL, 0
    MOVLW  HIGH(TAULAEUSART)
    MOVWF  TBLPTRH, 0
    MOVLW  UPPER(TAULAEUSART)
    MOVWF  TBLPTRU, 0
    RETURN
```

La funció *INIT_FLASH* l'únic que fa és agafar el valor LOW, HIGH i UPPER de l'etiqueta TAULAEUSART i posar-ho en els punters de la flash.

En total tenim 5 *INIT_FLASH* per a totes les etiquetes de la flash i el mateix amb *READ_FLASH* excepte l'etiqueta TAULAVALORSNOTES que no la volem mostrar.

```
TAULAEUSART
;MENU PRINCIPAL
DB "\n\rSelect an option: \n\r", "1 - New Game \n\r"
DB "2 - Resume Game\n\r"
DB "> ", "?"
TAULAEUSART_2
DB "\n\rSelect an option: \n\r", "1 - Global View \n\r"
DB "2 - Nuke\n\r", "3 - Songs\n\r"
DB "> ", "?"
TAULANOTES
DB "\n\rNotes: F4 F3 E5 E4 C4 B3 A3 D4 A4 G4\n\r", "> ?"
TAULAVALORSNOTES
DB .18,.37
DB .9,.19
DB .25,.26
DB .29,.22
DB .14,.16
TAULAENDGAME
DB "\n\rGUANYADOR JUGADOR: ?"
```

2.1.3. Lectura Opcions

Quan hem mostrat un menú d'opcions i volem llegir del RX si ens han seleccionat una opció, això ho fem amb 2 funcions que tenen la mateixa base, però un serveix per a llegir les opcions del menú 1 i l'altre del menú 2 (El menú 1 té les opcions New Game i start Game que es mostra a la fase 1 i al start game si no s'ha seleccionat cap opció prèviament, i el menú 2 és el que es mostra en tota la fase de joc per a seleccionar els opcionals).

```

READ_OPTION_1
    BTFSS    PIR1, RCIF,0
    RETURN
    MOVFF    RCREG,CONTINGUT
    MOVFF    CONTINGUT,VALUE
    CALL    MOSTRAR_CARACTER
    MOVLW  '\n'
    MOVWF   VALUE,0
    CALL    MOSTRAR_CARACTER
    MOVLW  '\r'
    MOVWF   VALUE,0
    CALL    MOSTRAR_CARACTER
    MOVLW .49
    SUBWF   CONTINGUT,0,0
    BTFSC   STATUS,Z,0
    SETF    FLAG_OPCIO_1,0
    MOVLW .50
    SUBWF   CONTINGUT,0,0
    BTFSC   STATUS,Z,0
    SETF    FLAG_OPCIO_2,0
    BTFSC   FLAG_OPCIO_1,0,0
    RETURN
    BTFSC   FLAG_OPCIO_2,0,0
    RETURN
    CALL    MOSTRAR_NOVA_PAG
    CALL    READ_FLASH
    RETURN

```

En la funció *READ_OPTION_1*, comprovem si tenim una dada disponible al registre **RCREG**, si és que si, el copiem al registre **CONTINGUT** i **VALUE** per mostrar l'opció introduïda, fem un salt de linea i llavors comprovem l'opció que ens han introduït, si és un 1, activem el flag **FLAG_OPCIO_1**, si val 2 activem el flag **FLAG_OPCIO_2**. (Més endavant explicarem com gestionem aquests flags)

Finalment, si no es cap de les anteriors, tornem a mostrar el menú perquè ens han introduït algun símbol erroni.

```

READ_OPTION_2
    BTFSS   PIR1, RCIF,0
    RETURN
    MOVFF   RCREG,CONTINGUT
    MOVFF   CONTINGUT,VALUE
    CALL  MOSTRAR_CARACTER
    MOVLW  '\n'
    MOVWF  VALUE,0
    CALL  MOSTRAR_CARACTER
    MOVLW  '\r'
    MOVWF  VALUE,0
    CALL  MOSTRAR_CARACTER
    MOVLW .49
    SUBWF  CONTINGUT,0,0
    BTFSC  STATUS,Z,0
    GOTO  GLOVAL_VIEW_OPTION
    MOVLW .50
    SUBWF  CONTINGUT,0,0
    BTFSC  STATUS,Z,0
    GOTO  NUKE_OPTION
    MOVLW .51
    SUBWF  CONTINGUT,0,0
    BTFSC  STATUS,Z,0
    GOTO  SONG_OPTION
    CALL  MOSTRAR_NOVA_PAG
    CALL  READ_FLASH_2
    RETURN

```

En la funció *READ_OPTION_2* fem el mateix, però en lloc d'activar flags, anem a la funció de l'opcional que s'ha seleccionat.

2.1.4 Mostratge matriu

Per a mostrar una matriu per la terminal, utilitzem diverses funcions, la principal és un bucle que es repeteix 64 cops i llegeix la RAM i envia en caràcter que toca cada cop, el funcionament és semblant a la funció de mostrar la matriu de leds explicada en la fase 2.

Funció LECTURA_VALORS_EU, primer comprovem de quin jugador volem mostrar el taulell amb el bit CURRENT_GRID, si val 0, FSROL començarà a l'adreça 0, sí, en canvi, val 1, carreguem un 64.

Llavors fem un salt de linea enviant '\n' i '\r' i comencem amb el bucle per enviar les 64 caselles del taulell.

```

743    LECTURA_VALORS_EU
744        CLRF FSROL, 0
745        MOVLW .64
746        BTFSC CURRENT_GRID,0,0
747        MOVWF FSROL,0
748
749        ;CLRF COMPT_EEPROM,0;
750
751        CLRF CANVI_LINEA,0      ;USART
752        MOVLW '\n'
753        MOVWF VALUE,0
754        CALL MOSTRAR_CARACTER
755        MOVLW '\r'
756        MOVWF VALUE,0
757        CALL MOSTRAR_CARACTER
758    BUCLE_LECTURA_EU

```

El primer que fem dintre del bucle és incrementar el registre CANVI_LINEA, aquesta variable la fem servir per comptar fins a 8 i fer un salt de linea cada 8 caselles mostrades.

Mentre estem dintre del bucle, el que fem és llegir el contingut de la RAM amb **POSTINCO** i guardar-la al registre **VALOR**, aquesta funció ens serveix per mostrar la matriu de 3 maneres diferents, en la fase 1 que només mostrem els vaixells que anem introduint, en la fase de joc que mostrem els atacs i el gloval view que mostra el vaixell ocult i els atacs.

Si volguéssim mostrar el Gloval View cridaríem la funció *ADRESS_TO_CHAR_2*, però si no, cridarem la funció *ADRESS_TO_CHAR*.

Comprovem si hem mostrat tot el taulell, si no, repetim el bucle, si ja hem mostrat tot el taulell anem a *END_LECTURA_RAM_EU* on fem un salt de linea i RETURN.

```

758    BUCLE_LECTURA_EU
759        INCF CANVI_LINEA,1,0
760
761        MOVFF POSTINCO, VALOR          ;Guardem la posicio de la RAM actual i incrementem el punter
762        ;Tenim el color de tots els taulells.
763        BTFSS GLOVAL_VIEW, 0, 0 ;SI L'USUARI HA COMENÇAT EL JOC MOSTRAREM ELS VALORS DE LA FASE 2
764        CALL ADRESS_TO_CHAR
765        BTFSC GLOVAL_VIEW, 0, 0 ;SI L'USUARI NO HA COMENÇAT EL JOC MOSTRAREM ELS VALORS DE LA FASE 1
766        CALL ADRESS_TO_CHAR_2
767        MOVLW .63
768        CPFSGT FSROL, 0           ;Jugador 0
769        GOTO BUCLE_LECTURA_EU
770        BTFSS CURRENT_GRID,0,0
771        GOTO END_LECTURA_RAM_EU
772        MOVLW .127
773        CPFSGT FSROL, 0           ;Jugador 0
774        GOTO BUCLE_LECTURA_EU
775
776        CLRF CANVI_LINEA,0
777        MOVLW '\n'
778        MOVWF VALUE,0
779        CALL MOSTRAR_CARACTER
780        MOVLW '\r'
781        MOVWF VALUE,0
782        CALL MOSTRAR_CARACTER
783
784    END_LECTURA_RAM_EU

```

Amb la funció *ADRESS_TO_CHAR* mostrem una casella, primer enviem el símbol '[' després comprovem si *STARTGAME* val 0 o 1 depenent si estem fase 1 o fase de joc i cridem *SW_VIEW* o *SW_VIEW_2* depenent del cas.

Aquestes funcions ens retornen el valor del símbol a enviar depenent si és aigua, vaixell, tocat o enfonsat depenent de la fase.

Finalment enviem '[' i comprovem si *CANVI_LINEA* val 8 per fer un salt de linea.

```

ADRESS_TO_CHAR
    MOVLW '['
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER
    BTFSS STARTGAME, 0, 0
    CALL SW_VIEW
    BTFSC STARTGAME, 0, 0
    CALL SW_VIEW_2
    CALL MOSTRAR_CARACTER
    MOVLW ']'
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER

    MOVLW .7
    CPFSGT CANVI_LINEA,0
    RETURN
    CLRF CANVI_LINEA,0
    MOVLW '\n'
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER
    MOVLW '\r'
    MOVWF VALUE,0
    GOTO MOSTRAR_CARACTER
    RETURN

```

La funció *SW_VIEW* el que fa és retornar '=' si és aigua i '=' si és vaixell.

```

SW_VIEW
    MOVLW '='
    MOVWF VALUE,0
    MOVLW .1
    CPFSLT VALOR,0
    RETURN
    MOVLW ' '
    MOVWF VALUE,0
    RETURN

```

La funció *SW_VIEW_2* retorna ' ' si és aigua o el vaixell no s'ha tocat, 'X' si ha fallat i '=' si ha encertat.

```
SW_VIEW_2
    MOVLW ' '
    MOVWF VALUE,0
    MOVLW .1
    CPFSGT VALOR,0
    RETURN
    MOVLW 'X'
    MOVWF VALUE,0
    MOVLW .2
    CPFSGT VALOR,0
    RETURN
    MOVLW '='
    MOVWF VALUE,0
    RETURN
```

La funció *SW_VIEW_3* és igual que la 2, però en el cas del vaixell ocult retorna '*'.

```
SW_VIEW_3
    MOVLW ' '
    MOVWF VALUE,0
    MOVLW .0
    CPFSGT VALOR,0
    RETURN
    MOVLW '**'
    MOVWF VALUE,0
    MOVLW .1
    CPFSGT VALOR,0
    RETURN
    MOVLW 'X'
    MOVWF VALUE,0
    MOVLW .2
    CPFSGT VALOR,0
    RETURN
    MOVLW '='
    MOVWF VALUE,0
    RETURN
```

2.1.5 Opcional 1.3: Menú Avançat Automàtic

En aquest opcional hem de mostrar el menú 1 quan estem introduint les coordenades dels vaixells i després de fer start game si no s'ha seleccionat cap de les 2 opcions, no comencem partida fins que seleccioni si és New Game o Resume Game.

El que fem és en el bucle de fase 1 on estem introduint els vaixells de la fase 1, anem consultant el RX constantment cridant la funció *READ_OPTION_1*, si se selecciona l'opció de Resum Game simplement, seguim guardant els vaixells , deixarem de mostrar el menú 1 i ja no llegirem més el RX per evitar que ens introduceixin un Resume Game.

```

2202      FASE1
2203      BTFSS FLAG_OPCIO_1,0,0
2204      CALL   READ_OPTION_1
2205      BTFSC FLAG_OPCIO_2,0,0
2206      GOTO  INIT_FASE2
2207      MOVLW .5
2208      CPFSLT COMPT_LED,0
2209      CALL   CONFIG_ADC_LED
2210      BTFSS  PORTD,RD2,0    ;ESPEREM EL SENYAL START GAME PER SABER SI
2211      GOTO  FASE1          ;HEM DE SORTIR DEL BUCLE DE LA FASE 1 I

```

En ser el menú avançat automàtic, només volem mostrar la matriu quan ens arribi el senyal New Boat que indica que ja han introduït un vaixell. Per això cridem la funció *LECTURA_VALORS_EU* per mostrar la matriu en el RSI de la interrupció INT1 que és la del New Coord, també comprovem si el **FLAG_OPCIO_1** està desactivat per mostrar un altre cop el menú 1.

```

RSI_INT1                                ;Interrupció del NewBoat per ca
                                         ;que ja han introduït un vaixell
                                         ;i activar el flag FLAG_OPCIO_1
                                         ;per a mostrar el menú 1
                                         ;en el bucle de la fase 1
                                         ;i saltar a la funció INIT_FASE2
                                         ;que ens carregará tot el que tenim guardat a
                                         ;l'EEPROM i inicialitzem tot el necessari, la interrupció, les variables...
                                         ;i el menú 1
                                         ;i deixarem de mostrar el menú 1 i ja no llegirem més el RX
                                         ;per evitar que ens introduceixin un Resume Game.

      CALL  MOSTRAR_NOVA_PAG
      CALL  LECTURA_VALORS_EU
      BTFSS  FLAG_OPCIO_1,0,0
      CALL  READ_FLASH

      BTG  CURRENT_GRID,0,0

```

En el cas que ens seleccionin l'opció 2, que és el Resum Game, activem el flag **FLAG_OPCIO_2** i en el bucle de la fase 1, saltarem a la funció **INIT_FASE_2** que ens carregarà tot el que tenim guardat a l'EEPROM i inicialitzem tot el necessari, la interrupció, les variables...

```

INIT_FASE2
    CALL CARREGAR_EEPROM
    ;CALL CONFIG_INT
    CLRF ACTUAL_PWM,0
    ;CALL INIT_TIMER0

    ;BCF
    BSF LATA,RA4,0
    BTFSC CURRENT_GRID,0,0
    BCF LATA,RA4,0
    SETF STARTGAME,0
    CLRF FSROL, 0

    MOVLW .1
    CPFSGT PWM_S0,0
    GOTO END_GAME
    CPFSGT PWM_S1,0
    GOTO END_GAME

    CALL CONVERT_COORD_TO_RAM
    CALL LECTURA_VALORS
    MOVLW .255
    MOVWF VALUEESPERA,0
    CALL ESPERA

    CALL LECTURA_VALORS_EU
    CALL READ_FLASH_2

    GOTO LOOP

```

Finalment, quan estem al LOOP de la fase 2, anem cridant la funció *READ_OPTION_2* per mirar si ens ha arribat alguna cosa pel RX.

```

CALL INICIALIZAR_EEPROM
LOOP ;fem polling dels pulsadors

    BTFSC PORTD,RD1,0 ;Esperem a que es premi el pulsador.
    CALL CONTROL_RIGHT ;Esperem a que es premi el pulsador.
    BTFSS PORTB,RB5,0 ;Esperem a que es premi el pulsador.
    CALL CONTROL_DOWN ;Esperem a que es premi el pulsador.
    BTFSC PORTD,RD3,0 ;Esperem a que es premi el pulsador.
    CALL CONTROL_ATTACK ;Esperem a que es premi el pulsador.
    BTFSS PORTB,RB4,0 ;Esperem a que es premi el pulsador.
    CALL CONTROL_LEFT ;Esperem a que es premi el pulsador.
    BTFSS PORTB,RB3,0 ;Esperem a que es premi el pulsador.
    CALL CONTROL_UP ;Esperem a que es premi el pulsador.
    CALL CONFIG_ADC_X

    MOVLW .5
    CPFSLT COMPT_LED,0
    CALL CONFIG_ADC_LED
    → CALL READ_OPTION_2
    GOTO LOOP

```

I mostrem la matriu cada cop que es fa un atac i tot seguit mostrem el Menú 2. Que els cridem en les funcions *HAS_TOCAT_AIGUA* i *HAS_TOCAT_BARCO*.

```

HAS_TOCAT_AIGUA
    ;POSAR LA POSICIO EN VERMELL
    MOVLW .2
    MOVWF INDF0, 0          ;POSEM 10 A LA COORDENADA ATACADA
    CALL GUARDAR_ATACK
    MOVLW b'00001001'         ;MOVEM EL CURSOR FORA DE LA MatriU A MOSTRAR
    MOVWF COORD_Y, 0
    ;MOSTREM LA MatriU ACTUALITZADA
    CLRF FSROL,0
    CALL LECTURA_VALORS
    CLRF FSROL,0
    CALL LECTURA_VALORS_EU
    CALL READ_FLASH_2

```

2.1.6 Extres

Perquè sigui més agradable visualment, cada cop que mostrem un taulell o fem un canvi significatiu, fem un salt de pàgina enviant el valor 12 en decimal, utilitzant la funció *MOSTRAR_NOVA_PAGA*

```

MOSTRAR_NOVA_PAG
    MOVLW .12
    MOVWF VALUE, 0
    CALL MOSTRAR_CARACTER
    RETURN

```

També, perquè sigui més fàcil saber quin jugador ha guanyat, hem implementat una funcionalitat que mostra per consola, quin jugador ha guanyat. Per a fer-ho, simplement comprovem la vida dels servos dels dos jugadors, és a dir, si el **PWM_S0** està a 0, voldrà dir que ha guanyat el jugador 1 i, per tant, cridem la funció *GUANYADOR_1*. Farem el mateix, en cas de guanyar el jugador 0, amb la seva respectiva funció *GUANYADOR_0*.

```

CALL MOSTRAR_NOVA_PAG
CALL READ_FLASH_5

```

```

MOVLW .1
CPFSGT PWM_S0,0
CALL GUANYADOR_1
MOVLW .1
CPFSGT PWM_S1,0
CALL GUANYADOR_0

```

```

GUANYADOR_1
    MOVLW '1'
    MOVWF VALUE, 0
    GOTO MOSTRAR_CARACTER

```

2.2 EEPROM

Un cop la EUSART ja va bé, hem plantejat com guardar tota la informació necessària a l'EEPROM.

Sabem que a l'EEPROM podem guardar 256 adreces, nosaltres les matrius dels 2 jugadors ens ocupen 128 adreces de la RAM, és a dir que sí que carreguem les 2 matrius ens sobre 128 adreces per guardar altra informació necessària per a la partida.

Primer pas, saber quina és la informació necessària per guardar, com guardar i llegir a l'EEPROM i adaptar-ho a la fase 2.

2.2.1 Distribució de la memòria

La distribució de la EEPROM que hem fet és la següent:

L'adreça 0 a 127 hi guardem els 2 taulells.

L'adreça 128, 129 i 130 les coordenades X i Y del cursor i el Current Grid per saber quin jugador estava jugant.

L'adreça 131 i 132 les vides dels 2 jugadors.

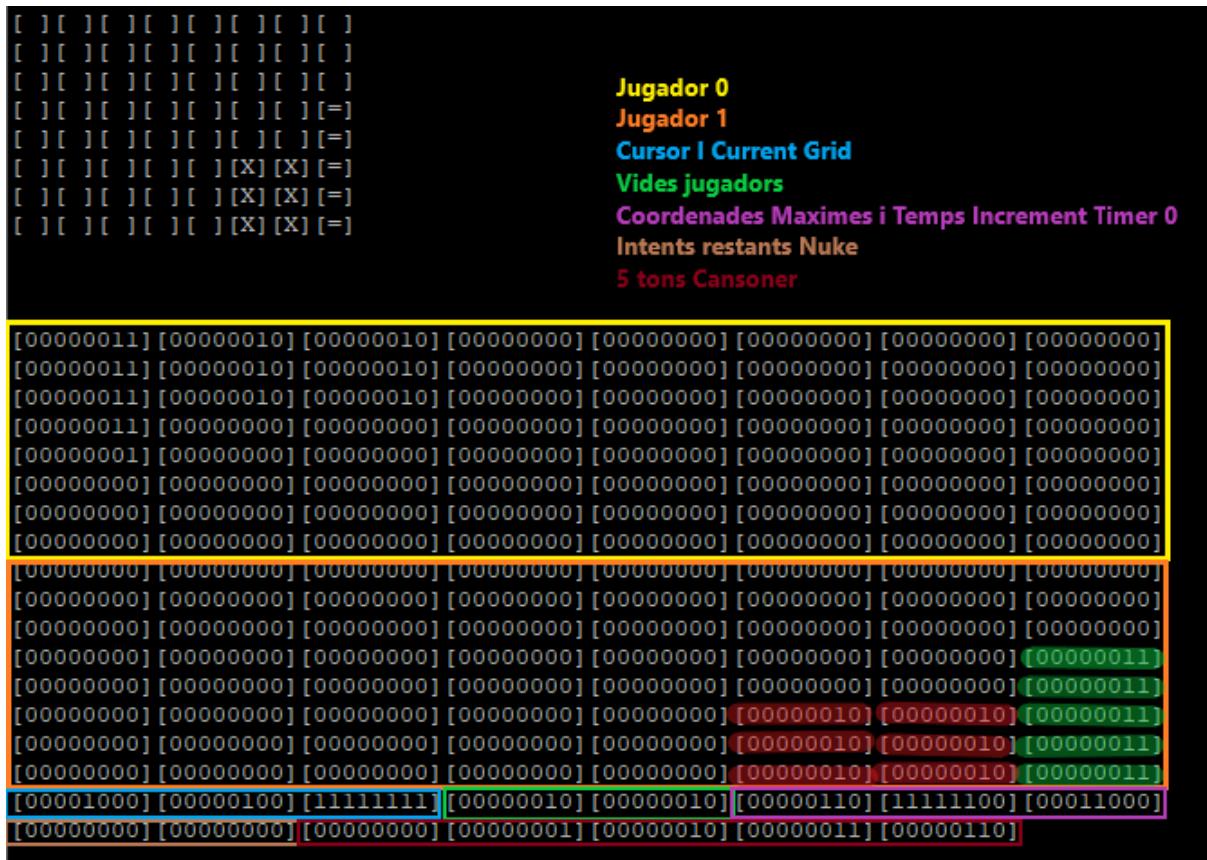
L'adreça 133 el nombre de caselles màxim de vaixells introduïts.

L'adreça 134 i 135 els temps del timer 0 de l'increment que toca depenent del nombre de caselles de vaixells introduïts.

L'adreça 136 i 137 els intents de tirar el Nuke que els hi queda a cadascun.

Finalment de l'adreça 138 a 143 les notes del cançoner.

Perquè ens sigui més còmoda a l'hora de guardar i llegir aquesta informació, hem decidit primer guardar-la a la RAM, ja que és molt més fàcil de manipular, llavors fer un bucle per passar les 143 adreces de la RAM a l'EPROM.



2.2.2 Lectura i escriptura de l'EEPROM

Hem creat 2 funcions per llegir i escriure a l'EEPROM.

Funció **LECTURA_EEPROM**, movem el valor de **FSROL** a l'**EEADR** per indicar-li a quina adreça volem llegir, inicialment valdrà 0 i llavors amb el registre **EECON1** indiquem que volem llegir l'EEPROM.

Tot seguit el resultat el tindrem a **EEDATA**, el passem a la RAM amb **POSTINCO** per incrementar l'adreça, tot seguit comprovem el valor de **FSROL**, si val 143 vol dir que ja hem acabat de llegir tots els valors de l'EEPROM i acaben, si no tornem a fer la lectura.

```

LECTURA_EEPROM
    MOVF FSROL,0 ;

    MOVWF EEADR,0 ; Data Memory Address to write

    BCF EECON1, EEPGD,0 ; Point to DATA memory
    BCF EECON1, CFGS,0 ; Access EEPROM
    BSF EECON1, RD,0 ; EEPROM Read
    MOVFF EEDATA, POSTINCO ; W = EEDATA
;MOVlw .136
    MOVlw .143
    CPFSEQ FSROL, 0           ;Jugadc
    GOTO LECTURA_EEPROM
    CLRF FSROL,0
    RETURN

```

Per a l'escriptura de l'EEPROM ho hem fet en varíes funcions, primer hem creat la funció bàsica per a escriure que és *PROCES_ESCRIPTURA_EEPROM* , el que fem aquí és en el registre EECON1 l'hi indiquem que volem escriure, llavors desabilitem la interrupció i fem una seqüència que s'ha de fer d'aquesta manera per seguretat.

Després tornem a habilitar la interrupció i ens esperem que el bit WR del registre **EECON1** valgui 0 que indica que ha finalitzat l'escriptura.

Abans de cridar aquesta funció hem de guardar al registre **EEADR** l'adreça que volem modificar i a **EEDATA** la dada que volem guardar.

```

PROCES_ESCRIPTURA_EEPROM
    BCF EECON1, EEPGD,0 ; Point to DATA memory
    BCF EECON1, CFGS,0 ; Access EEPROM
    BSF EECON1, WREN,0 ; Enable writes

    BCF INTCON, GIE ; Disable Interrupts
    MOVlw 55h ;
    MOVWF EECON2 ; Write 55h
    MOVlw 0AAh ;
    MOVWF EECON2 ; Write 0AAh
    BSF EECON1, WR ; Set WR bit to begin write

    BSF INTCON, GIE, 0 ; Enable Interrupts

    ESPERA_END_E
    BTFSC EECON1,WR,0
    GOTO ESPERA_END_E

    RETURN

```

Hem dividit en 2 parts l'escriptura de la EEPROM, la primera és quan hem acabat fase 1 i volem començar fase 2 amb New Game, guardem de cop tota la informació de cop a la EEPROM, els 143 valors de la RAM els carreguem a l'EEPROM.

Llavors un cop inicialitzada l'EEPROM, en la segona part que és en la fase de joc, només modifiquem les adreces necessàries, per exemple si fem un atac al jugador 0 en la posició 2,3 , només modificarem aquesta adreça de l'EEPROM.

La funció *INICIALIZAR_EEPROM* és la que carrega les 143 adreces de la RAM a l'EEPROM.

Fa el procés contrari que *LECTURA_EEPROM* , carreguem el valor de FSROL a EEADR, i llavors la dada de la RAM a EEDATA amb el POSTINCO, tot seguit cridem a la funció *PROCES_ESCRIPTURA_EEPROM* que ens escriurà a l'EEPROM, això ho fem 143 cops.

```
INICIALIZAR_EEPROM
    ;INCF CANVI_LINEA,1,0
    MOVF FSROL,0
    MOVWF EEADR,0 ; Data Memory Address to write
    MOVF POSTINCO,0 ;
    ;CALL ADRESS_TO_CHAR
    MOVWF EEDATA,0 ; Data Memory Value to write

    CALL PROCES_ESCRIPTURA_EEPROM

    MOVLW .143
    CPFSEQ FSROL, 0                                ;Jugado
    GOTO INICIALIZAR_EEPROM
    CLRF FSROL,0
    RETURN
```

Durant la fase de joc tenim les següents funcions que van actualitzant la EEPROM quan toca.

Són *GUARDAR_CURSOR*, *GUARDAR_VIDES*, *GUARDAR_INFO_SERVOS*, *GUARDAR_ATACK*, *GUARDAR_NUKE* i *GUARDAR_SONG*.

Totes són similars amb poques diferències.

Per exemple a *GUARDAR_CURSOR*, carreguem el valor 128 a **EEADR** i el contingut del registre **COORD_X** a **EEDATA** i cridem la funció *PROCES_ESCRIPTURA_EEPROM*, fem el mateix amb **COORD_Y** i **CURRENT_GRID** però amb les adreçess 129 i 130 respectivament.

```

GUARDAR_CURSOR
    MOVLW .128
    MOVWF EEADR,0 ; Data Memory Address to write
    MOVFF COORD_X,EEDATA
    CALL PROCES_ESCRIPTURA_EEPROM
    MOVLW .129
    MOVWF EEADR,0 ; Data Memory Address to write
    MOVFF COORD_Y,EEDATA
    CALL PROCES_ESCRIPTURA_EEPROM
    MOVLW .130
    MOVWF EEADR,0 ; Data Memory Address to write
    CLRF EEDATA,0
    BTFSC CURRENT_GRID,0,0
    SETF EEDATA,0
    CALL PROCES_ESCRIPTURA_EEPROM
    RETURN

```

GUARDAR_ATACK, la cridem tot just després de fer un atac i com que guardem el valor a la RAM amb **INDFO**, l'adreça al **FSROL** no s'haurà modificat, així que guardem aquesta a **EEADR** i el contingut de **INDFO** a **EEDATA** i cridem *PROCES_ESCRIPTURA_EEPROM*.

```

GUARDAR_ATACK
    MOVFF FSROL, EEADR ; Data Memory Address to write
    MOVFF INDFO,EEDATA
    CALL PROCES_ESCRIPTURA_EEPROM

    RETURN

```

2.2.3 DEBUG EEPROM

Hem creat una funció per a mostrar el contingut de la EEPROM a la pantalla del Putty, d'aquesta manera podem veure si guardem bé, i tot el que està passant en la EEPROM en tot moment.

La funció *LECTURA_VALORS_EU_EP*, és la que mostra tota la EEPROM pel terminal, el que fem al començament és iniciar a 0 el comptador **COMPT_EEPROM**, i **CANVI_LINEA**.

Fem un salt de linea i comencem amb el bucle de lectura, incrementem el comptador de **CANVI_LINEA** que quan arriba a 8 fem un salt de linea.

Movem el valor de **COMPT_EEPROM** a l'**EEADR** i fem el procés de lectura, llavors movem el contingut de **EEDATA** a **VALUE_E** i cridem la funció *L_EEPROM* que ens enviarà aquesta informació per la EUSART.

Finalment, comprovem si el comptador **COMPT_EEPROM** val 143 per acabar el bucle.

```
801    LECTURA_VALORS_EU_EP
802        CLRF FSROL, 0
803        CLRF COMPT_EEPROM, 0;
804
805        CLRF CANVI_LINEA, 0      ;USART
806        MOVLW '\n'
807        MOVWF VALUE, 0
808        CALL MOSTRAR_CARACTER
809        MOVLW '\r'
810        MOVWF VALUE, 0
811        CALL MOSTRAR_CARACTER
812    BUCLE_LECTURA_EU_EP
813        INCF CANVI_LINEA, 1, 0
814
815        MOVF COMPT_EEPROM, 0 ;
816
817        MOVWF EEADR, 0 ; Data Memory Address to write
818
819        BCF EECON1, EEPGD, 0 ; Point to DATA memory
820        BCF EECON1, CFGS, 0 ; Access EEPROM
821        BSF EECON1, RD, 0 ; EEPROM Read
822
823        MOVFF EEDATA, VALUE_E ; W = EEDATA
824
825        CALL L_EEPROM
826
827        MOVLW .7
828        CPFSGT CANVI_LINEA, 0
```

```

829      GOTO SKIP_L
830      CLRF CANVI_LINEA,0
831      MOVLW '\n'
832      MOVWF VALUE,0
833      CALL MOSTRAR_CARACTER
834      MOVLW '\r'
835      MOVWF VALUE,0
836      CALL MOSTRAR_CARACTER
837
838      SKIP_L
839
840      INCF COMPT_EEPROM,1,0
841
842      MOVLW .143
843      CPFSEQ COMPT_EEPROM, 0
844      GOTO BUCLE_LECTURA_EU_EP
845      END_LECTURA_RAM_EU_EP
846      MOVLW '\n'
847      MOVWF VALUE,0
848      CALL MOSTRAR_CARACTER
849      MOVLW '\r'
850      MOVWF VALUE,0
851      CALL MOSTRAR_CARACTER
852      CLRF FSROL, 0
853      RETURN

```

La funció L_EEPROM és la que rep la dada de l'EEPROM que volem enviar i l'envia per la EUSART per mostrar-la en format binari.

El funcionament bàsic és similar a la funció PRINTAR_LED de la fase 2.

El que fem al començament és enviar un '[', llavors comencem el bucle que es farà 8 cops perquè els registres són de 8 bits. Carreguem un '0' al **WREG**, llavors comprovem el LSB del registre **VALUE_E** si val 0 amb un BTFSC, si val 0 saltem la següent línia que és carregar un '1' al **WREG**.

Després fem un rotate a l'esquerra a **VALUE_E** i **COMPTADOR** que està inicialitzat a b'10000000'.

Després carreguem aquest '0' o '1' a **VALUE** i el mostrem, tot seguit comprovem el MSB de **COMPTADOR**, si val 1 vol dir que ja l'hi hem fet 8 rotates hem acabat i mostrem ']' per finalitzar, si no tornem a fer el bucle.

```

L_EEPROM
    MOVLW '['
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER

BUCLE_E
    MOVLW '0'

    BTFSC VALUE_E,7,0
    MOVLW '1'
    rlncf VALUE_E,1,0
    rlncf COMPTADOR,1,0
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER

    BTFSS COMPTADOR,0,0
    GOTO BUCLE_E
    MOVLW ']'
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER

RETURN

```

2.2.4 Opcional 2: Global View

Per a fer l'opció de Global View, dins de la funció *READ_OPTION_2*, durant l'execució de la fase 2, tindrem l'opció de Global View disponible, que una vegada seleccionada, anirà a la funció *GLOVAL_VIEW_OPTION*.

Un cop a dins, posarem a 1 la variable **GLOVAL_VIEW**, per indicar que estem executant aquesta opció. Seguidament, farem un BTG de **CURRENT_GRID**, per a poder mostrar el nostre taulell en l'estat actual i després farem una crida a la funció *LECTURA_VALORS_EU*, que mostrerà la matriu pel terminal. Després només caldrà tornar a fer BTG de **CURRENT_GRID**, posar a 0 **GLOVAL_VIEW** i tornar a mostrar el menú d'opcions.

```

GLOVAL_VIEW_OPTION
    SETF GLOVAL_VIEW,0
    BTG CURRENT_GRID,0,0
    CALL LECTURA_VALORS_EU
    BTG CURRENT_GRID,0,0
    CLRF GLOVAL_VIEW,0
    CALL READ_FLASH_2
RETURN

```

Dintre de la funció *LECTURA_VALORS_EU*, en tenir ara la variable **GLOVAL_VIEW** a 1, seleccionarem la funció *ADRESS_TO_CHAR_2*, que és idèntica que la *ADRESS_TO_CHAR* però amb l'única diferència que crida la funció *SW_VIEW_3* que s'encarrega de convertir els valors guardats en la RAM a símbols del Gloval View.

```

ADRESS_TO_CHAR_2
    MOVLW '['
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER
    CALL SW_VIEW_3
    CALL MOSTRAR_CARACTER
    MOVLW ']'
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER

    MOVLW .7
    CPFSGT CANVI_LINEA,0
    RETURN
    CLRF CANVI_LINEA,0
    MOVLW '\n'
    MOVWF VALUE,0
    CALL MOSTRAR_CARACTER
    MOVLW '\r'
    MOVWF VALUE,0
    GOTO MOSTRAR_CARACTER
    RETURN

```

En la funció *SW_VIEW_3*, mostrem ‘ ‘ si és aigua, ‘*’ si és casella de vaixell sense tocar, ‘=’ casella tocada i ‘X’ tocat aigua.

```

SW_VIEW_3
    MOVLW ' '
    MOVWF VALUE,0
    MOVLW .0
    CPFSGT VALOR,0
    RETURN
    MOVLW '*'
    MOVWF VALUE,0
    MOVLW .1
    CPFSGT VALOR,0
    RETURN
    MOVLW 'X'
    MOVWF VALUE,0
    MOVLW .2
    CPFSGT VALOR,0
    RETURN
    MOVLW '='
    MOVWF VALUE,0
    RETURN

```

2.3 ADC

Per a poder implementar el joystick i l'opcional per regular la intensitat lumínica de la matriu de leds amb el potenciómetre, hem hagut d'implementar la part de ADC on rebrem valors analògics que nosaltres podrem interpretar per, en aquest cas, moure el cursor i regular la llum de la matriu de leds.

2.3.1 Joystick

Per a implementar el moviment del cursor de la matriu, hem utilitzat un joystick, el qual està format per dos potenciómetres, un per a l'eix Y i un altre per a l'eix X. Per a poder veure si estem fent un moviment del cursor, hem creat dues funcions les quals ens serviran per a fer el polling del valor analògic tant de l'eix X com el Y.

Primer de tot, per a l'eix X, el que farem és configurar el registre **ADCON0**, habilitant la conversió del valor analògic posant a 1, el bit 0 i també indicant que agafarem el valor del port RA0.

A continuació es crida la funció **POLLING_GODONE**, en la que esperarem que la conversió s'acabi de fer i quan el bit 0 del **ADCON0** torni a ser 0 retornarem a l'anterior funció.

Finalment, agafarem el valor de la conversió AD del registre **ADRESH** i el guardarem a la variable **JOYX** en aquest cas. Després simplement cridarem la funció encarregada de realitzar el moviment de l'eix X.

Tot el dit fins ara serà idèntic per a l'eix Y, on només canviarà la configuració del registre **ADCON0** on indicarem el canal de lectura RA1 i la variable **JOYY** on guardarem el resultat.

```

CONFIG_ADC_X
    MOVLW  b'00000001'
    MOVWF  ADCON0,0
    CALL   ESPERA_16MS
    BSF    ADCON0,1,0
    CALL   POLLING_GODONE
    MOVFF  ADRESH,JOYX
    CALL   MOURE_JOYX
    RETURN

CONFIG_ADC_Y
    MOVLW  b'00000101'
    MOVWF  ADCON0,0
    CALL   ESPERA_16MS
    BSF    ADCON0,1,0
    CALL   POLLING_GODONE
    MOVFF  ADRESH,JOYY
    CALL   MOURE_JOYY
    RETURN

POLLING_GODONE
    BTFSC  ADCON0,1
    GOTO   POLLING_GODONE
    RETURN

```

Per a realitzar l'execució del moviment del cursor, un cop obtenim el valor de **JOYX**, cridarem la funció **MOURE_JOYX** on compararem el valor de **JOYX** amb tres rangs que determinaran si anem cap

a la dreta, esquerra en aquest cas o bé ens mantenim al mig. En cas d'anar a la dreta o esquerra cridarem les respectives funcions i si no és així, llavors cridem la funció *CONFIG_ADC_Y*, el qual després entraria a la funció *MOURE_JOYY* on repetiríem el mateix procediment que l'eix X, l'únic que ara anant amunt o avall.

```

MOURE_JOYX
;Em de saber si es dreta o esquerra
;Si es dreta CALL MOURE_RIGHT
;Si es esquerra CALL MOURE_LEFT

MOVLW .66
CPFSGT JOYX,0
GOTO LEFT
MOVLW .132
CPFSGT JOYX,0
GOTO CONFIG_ADC_Y
MOVLW .194
CPFSLT JOYX,0
GOTO RIGHT
GOTO CONFIG_ADC_Y

MOURE_JOYY
;Em de saber si es up o down
;Si es up CALL MOURE_UP
;Si es down CALL MOURE_DOWN

MOVLW .66
CPFSGT JOYY,0
GOTO UP
MOVLW .132
CPFSGT JOYY,0
GOTO MIG_Y
MOVLW .198
CPFSLT JOYY,0
GOTO DOWN
GOTO MIG_Y

```

Ara bé, si nosaltres no haguéssim mogut el joystick, llavors el resultat seria la crida de la funció *MIG_Y* la qual s'encarregaria de posar a 0 el *JOY_FLAG_Y*, que aquesta és la nostra variable de control, per saber si hem mogut o no el cursor.

En cas que per exemple haguéssim dit que volem moure el cursor a l'esquerra, miraríem si el flag està a 0, esperaríem 16 ms i llavors faríem el moviment del cursor (implementat ja a la fase 2). Un cop fet el moviment, llavors posaríem el flag a 1, de manera que si volgués fer un altre moviment, com amunt o avall, no podria, ja que a l'haver fet un moviment ja, no es mouria més. D'aquesta manera evitem que el cursor es mogués en diagonal, i en fer que només en tornar al mig sigui quan el flag torna a valdre 0 evitem falsos moviments.

```

MIG_Y
    CLRF    JOY_FLAG_Y,0
    RETURN

LEFT
    MOVLW .1
    CPFSLT  JOY_FLAG_Y,0
    RETURN
    CALL    ESPERA_16MS
    CALL    MOURE_LEFT
;CALL    ESPERA_16MS
    SETF    JOY_FLAG_Y,0
    RETURN

```

2.3.2 Opcional 3: Brillantor de la pantalla variable

Per a la realització de la brillantor de la pantalla variable, nosaltres farem ús d'un potenciómetre, ja que ens permetrà regular amb precisió la intensitat lumínica de la matriu de leds.

Primer de tot el que farem és configurar el **ADCON0**, de manera que habilitem la conversió del port AN2. Seguidament, farem una crida de la funció *POLLING_GODONE*, per a saber si ja s'ha acabat la conversió del valor i finalment guardarem el valor en una variable anomenada **LED_INIT**. Aquesta serà la variable que ens indicarà amb quanta intensitat hem de mostrar la matriu.

A continuació posarem a 0 la variable **COMPT_LED**, que és l'encarregada de fer que entrem a la funció *CONFIG_ADC_LED* cada 100 ms en totes les parts del codi.

Després el que fem és carregar el valor de **LED_INT** al registre **WREG** i comprovar si hem arribat al final de la partida (si hi hem arribat, simplement fem un RETURN). En cas que no hagim arribat, mirarem el valor del registre **LED_INT_ANT**, que conté el valor antic de la lluminositat de la matriu, i mirar si és igual al valor de **LED_INT** (carregat ja al registre WREG).

En cas de ser iguals els valors guardem el valor de **LED_INT** a **LED_INT_ANT**, i si són diferents, llavors haurem de pintar la matriu amb la nova intensitat lumínica cridant la funció *LECTURA_VALORS*, tot utilitzant el valor de **LED_INT** i després actualitzarem el valor de **LED_INT_ANT**.

```

CONFIG_ADC_LED
    MOVLW  b'00001001'
    MOVWF  ADCONO,0
    CALL   POLLING_GODONE
    MOVFF  ADRESH,LED_INT

    CLRF  COMPT_LED,0
    MOVF  LED_INT,0
    BTFSC END_GAME_M,0,0
    RETURN
    CPFSEQ LED_INT_ANT,0
    CALL   LECTURA_VALORS
    MOVFF  LED_INT,LED_INT_ANT
    RETURN

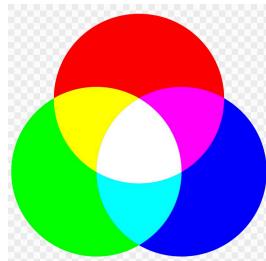
```

2.4 Opcional 4: Rainbow

Una vegada hagim finalitzat la partida, és a dir, que un dels dos jugadors guanyi, a part de la melodia del final del joc, haurem de mostrar una seqüència colorada de manera que es vagi mostrant en paral·lel a la cançó.

Per a fer això, hem implementat diversos colors i amb l'ajuda de la funció de pintar la matriu de la fase 2, hem pogut fer el rainbow.

Primer de tot, tenim les funcions que permetran mostrar els diferents colors de la matriu, que formaran el rainbow. Cal dir, però, que per a poder fer que la brillantor de la pantalla variable continues essent funcional, ens ha limitat el nombre de colors possibles, fent que només poguéssim utilitzar 7 colors diferents, a més del “negre” on apaguem els valors de la matriu per a formar-lo.



La funció per a pintar un color, és idèntica a la que ja varem implementar a la fase 2, on l'única variació ha estat que, en comptes d'assignar un valor determinat d'1's i 0's a un dels 3 colors, directament assignem la variable **LED_INT** a la component que corresponga amb el color que volem obtenir. En el cas del vermell, només ho hem assignat a la component Red.

```
ASSIGN_RED_RAINBOW
    MOVF LED_INT,0
    MOVWF LED_Red_Mirall,0
    CLRF LED_Blue_Mirall,0
    CLRF LED_Green_Mirall,0
    GOTO PRINT_LED
```

Ara bé, per a poder pintar la matriu de la manera que nosaltres vulguem hem de seguir diversos patrons, perquè a l'hora de pintar la matriu es vegin bé els colors i el moviment sigui fluid. En el nostre cas hem optat per a pintar 8 línies dels diferents 8 colors dels quals disposem, de manera que els anem rotant fins a 6 vegades diferents durant la finalització del joc. A continuació explicarem la lògica que hem utilitzat.

Primer de tot, des de la funció **END_GAME**, inicialitzarem les variables **COMPT_RAINBOW** i **COMPT_RAINBOW_R**, en el cas de la primera crida a ‘00000001’. Això serà la nostra manera d'assignar els diferents colors en ordres diferents, i per cada crida de les 6 que farem, inicialitzarem el valor de manera diferent. Seguidament, farem la crida de **PINTAR_RAINBOW**, la qual només entrar farà un RLNCF de la variable **COMPT_RAINBOW**. Tot seguit, comprovarem si hem arribat al final de la matriu, és a dir, si hem pintat tota la matriu, amb la variable **VALOR_RAINBOW**, la qual sempre a

L'inici de pintar una seqüència valdrà 0. En cas que no hagim acabat de pintar, cridarem la funció *BUCLE_PINTAR_RAINBOW*.

```
MOVlw b'00000001'
MOVwf COMPT_RAINBOW,0
MOVwf COMPT_RAINBOW_R,0
CALL PINTAR_RAINBOW
```

```
PINTAR_RAINBOW
    RLNCF COMPT_RAINBOW,1,0

    MOVLW .63
    CPFSGT VALOR_RAINBOW,0
    GOTO BUCLE_PINTAR_RAINBOW

    CLRF VALOR_RAINBOW,0
    RETURN
```

Una vegada dins de *BUCLE_PINTAR_RAINBOW*, incrementarem en una unitat la variable de **VALOR_RAINBOW**, ja que haurem pintat una casella. Seguidament, rotarem el bit de **COLOR_RAINBOW**, que serà l'encarregat de dir si hem acabat de pintar la línia actual o no, i cridarem la funció *ASIGN_RAINBOW*, que serà la que definirà el color amb què hem de pintar la línia, depenent del valor de **COMPT_RAINBOW**.

Un cop fet això, comprovarrem que no ens hagi saltat cap interrupció mirant el valor de la variable **MATRIU_CONTROL**, i en cas que haguessin saltat, aniríem a la funció *ESPERA_RESET_MATRIU_R*, la qual posaria a 0 la variable **VALOR_RAINBOW**, per a tornar a començar a pintar la matriu, faria una espera de 200us i coparia el valor de **COMPT_RAINBOW_R** (que és el nostre backup de la variable) a **COMPT_RAINBOW**.

```
ESPERA_RESET_MATRIU_R
    CLRF VALOR_RAINBOW, 0
    CALL ESPERA_100MICS
    CALL ESPERA_100MICS
    MOVFF COMPT_RAINBOW_R, COMPT_RAINBOW
```

Finalment, en cas que no hagués saltat cap interrupció, simplement aniríem pintant la línia i cada 8 caselles, canviant de color fins que haguéssim pintat les 64 caselles de la matriu i per tant acabat de pintar la matriu.

```
BUCLE_PINTAR_RAINBOW
SETF MATRIU_CONTROL,0
;RLNCF COMPT_RAINBOW,1,0
INCF VALOR_RAINBOW,1,0
RLNCF COLOR_RAINBOW,1,0
CALL ASIGN_RAINBOW

BTFS S MATRIU_CONTROL, 0, 0
GOTO ESPERA_RESET_MATRIU_R
BTFS COLOR_RAINBOW,0,0
GOTO BUCLE_PINTAR_RAINBOW
GOTO PINTAR_RAINBOW

ASIGN_RAINBOW
MOVLW b'00000001'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_RED
MOVLW b'00000010'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_YELLOW_RAINBOW
MOVLW b'000000100'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_GREEN
MOVLW b'00001000'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_PURPLE_RAINBOW
MOVLW b'00010000'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_BLUE
MOVLW b'00100000'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_CIAN_RAINBOW
MOVLW b'01000000'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_WHITE
MOVLW b'10000000'
CPFSGT COMPT_RAINBOW,0
GOTO ASSIGN_BLACK_RAINBOW
RETURN
```

2.5 Opcional 5: Nuke

Quan en el menú d'opcions ens seleccionen el Nuke, anem a la funció *NUKE_OPTION*.

El que fem primer a dintre de la funció és mirar qui jugador està jugant ara mateix amb el *CURRENT_GRID* i llavors comprovar si té intents disponibles, pel jugador 0 els seus intents estaran a **NUKE_CHECK_0** i l'altre a **NUKE_CHECK_1**.

Si no l'hi queden intents anem al final de la funció, sí, en canvi, sí que té intents, procedim a mostrar el cursor '*>*' i esperar que ens entrin els 2 valors de la coordenada a atacar.

```

1126    NUKE_OPTION
1127
1128        BTFSC CURRENT_GRID ,0,0
1129        MOVF NUKE_CHECK_0,0
1130        BTFSS CURRENT_GRID ,0,0
1131        MOVF NUKE_CHECK_1,0
1132
1133        MOVWF NUKE_TRYS,0
1134        BTFSS NUKE_TRYS ,0,0
1135        GOTO END_NUKE
1136
1137        BTFSC CURRENT_GRID ,0,0
1138        BCF NUKE_CHECK_0,0,0
1139        BTFSS CURRENT_GRID ,0,0
1140        BCF NUKE_CHECK_1,0,0
1141
1142        CALL GUARDAR_NUKE
1143        MOVLW '\n'
1144        MOVWF VALUE,0
1145        CALL MOSTRAR_CARACTER
1146        MOVLW '\r'
1147        MOVWF VALUE,0
1148        CALL MOSTRAR_CARACTER
1149        MOVLW '>'
1150        MOVWF VALUE,0
1151        CALL MOSTRAR_CARACTER
1152        MOVLW ' '
1153        MOVWF VALUE,0
1154        CALL MOSTRAR_CARACTER

```

Tenim dos bucles d'espera que el que fan és esperar a rebre dada pel RX, mentre estem a l'espera, anem monitorant el potenciómetre de la intensitat de la llum per anar-la actualitzant.

Quan ens introduceixen un valor, com que el rebem en ASCII, l'hi restem 48 en decimal com que és el 0 i a partir d'aquest estan els altres números en ordre.

```
1155    ESPERA_NUKE_1 |
1156        MOVLW .5
1157        CPFSLT COMPT_LED,0
1158        CALL CONFIG_ADC_LED
1159
1160        BTFSS PIR1, RCIF,0
1161        GOTO ESPERA_NUKE_1
1162        MOVFF RCREG,CONTINGUT
1163        MOVFF CONTINGUT,VALUE
1164        MOVFF VALUE,COORD_X
1165        MOVLW .48
1166        SUBWF COORD_X,1,0
1167        CALL MOSTRAR_CARACTER
1168        MOVLW ','
1169        MOVWF VALUE,0
1170        CALL MOSTRAR_CARACTER
1171    ESPERA_NUKE_2
1172        MOVLW .5
1173        CPFSLT COMPT_LED,0
1174        CALL CONFIG_ADC_LED
1175
1176        BTFSS PIR1, RCIF,0
1177        GOTO ESPERA_NUKE_2
1178        MOVFF RCREG,CONTINGUT
1179        MOVFF CONTINGUT,VALUE
1180        MOVFF VALUE,COORD_Y
1181        MOVLW .48
1182        SUBWF COORD_Y,1,0
1183        CALL MOSTRAR_CARACTER
```

Després de rebre les 2 coordenades, procedim a fer la seqüència d'atac, primer fem un BCF al bit 7 de **NUKE_TRYs** que ens servirà per saber si hem tocat algun vaixell.

Cridem la funció *ATACAR_NUKE* per fer l'atac a aquesta casella introduïda, després d'això donem la volta a aquesta coordenada i cada cop que ens movem fem un atac.

```

1192      BCF NUKE_TRYS,7,0
1193      CALL ATACAR_NUKE
1194
1195      DECFSZ COORD_Y, 1, 0      ;RESTEM 1 AL VALOR DE COORD_Y
1196      CALL ATACAR_NUKE
1197
1198      DECFSZ COORD_X, 1, 0      ;RESTEM 1 AL VALOR DE COORD_X
1199      CALL ATACAR_NUKE      ;TORNEM SI NO ESTEM A 1
1200
1201      MOVLW .1
1202      ADDWF COORD_Y, 1,0      ;SUMEM 1 AL VALOR DE LA COORD_Y
1203      CALL ATACAR_NUKE
1204
1205      MOVLW .1
1206      ADDWF COORD_Y, 1,0      ;SUMEM 1 AL VALOR DE LA COORD_Y
1207      CALL ATACAR_NUKE
1208
1209      MOVLW .1
1210      ADDWF COORD_X, 1,0      ;SUMEM 1 AL VALOR DE LA COORD_Y
1211      CALL ATACAR_NUKE
1212
1213      MOVLW .1
1214      ADDWF COORD_X, 1,0      ;SUMEM 1 AL VALOR DE LA COORD_Y
1215      CALL ATACAR_NUKE
1216
1217      DECFSZ COORD_Y, 1, 0      ;RESTEM 1 AL VALOR DE COORD_Y
1218      CALL ATACAR_NUKE
1219

```

Després de fer la seqüència d'atacs, mostrem el resultat en la matriu i en la pantalla, comprovem el bit 7 de **NUKE_TRYS** que ens indicarà si hem tocat algun vaixell per saber quin to de soroll fer.

Procedim a fer el to durant 1 segon i després 2 segons sense fer res, comprovem si hem acabat partida o no hi canviem de jugador i actualitzem la matriu.

Finalment, tornem a mostrar la matriu en la pantalla i mostrar el menú d'opcions.

```

1220      DECFSZ COORD_Y, 1, 0 ;RESTEM 1 AL VALOR DE COORD_Y
1221      CALL ATACAR_NUKE
1222
1223      MOVLW b'00001001'          ;MOVEM EL CURSOR FORA DE LA MatriU A MOSTRAR
1224      MOVWF COORD_Y,
1225      ;MOSTREM LA MatriU ACTUALITZADA
1226      CLRF FSROL,0
1227      CALL LECTURA_VALORS
1228
1229      CALL LECTURA_VALORS_EU
1230      CALL GUARDAR_CURSOR
1231
1232      CLRF COMPT_1S,0
1233
1234      MOVLW .78
1235      BTFSC NUKE_TRYS,7,0
1236      MOVLW .10
1237
1238      MOVWF SOROLL_FREQ,0
1239      CALL SOROLL_ALTAVEU
1240      CLRF COMPT_1S,0
1241      CALL ESPERA_1S
1242      CLRF COMPT_1S,0
1243      CALL ESPERA_1S
1244      MOVLW .1
1245      CPFSGT PWM_S0,0
1246      GOTO END_GAME
1247      CPFSGT PWM_S1,0
1248      GOTO END_GAME
1249
1250      CALL INIT_CURSOR
1251      BTG CURRENT_GRID,0,0
1252      BTG LATA,RA4,0
1253      CALL GUARDAR_CURSOR
1254
1255
1256      CLRF FSROL,0
1257
1258      CALL LECTURA_VALORS
1259
1260      END_NUKE
1261
1262      CALL LECTURA_VALORS_EU
1263      CALL READ_FLASH_2
1264
1265      RETURN
1266

```

La funció **ATACAR_NUKE** és la que fa l'atac, abans comprovant que el cursor segueix dintre de la regió d'atac, comprovant la **COORD_X** i **COORD_Y** que no hagin sortit dels marges, si només un d'ells ha sortit del marge, no fem l'atac i tornem.

```

1267    ATACAR_NUKE
1268        MOVLW .9
1269        CPFSLT COORD_X, 0      ;COMPARA EL WREG AMB COORD_X, SI COORD_Y ES MES GRAN SALTA DE LINEA
1270        RETURN
1271
1272        MOVLW .9
1273        CPFSLT COORD_Y, 0      ;COMPARA EL WREG AMB COORD_X, SI COORD_Y ES MES GRAN SALTA DE LINEA
1274        RETURN
1275
1276        MOVLW .0
1277        CPFSGT COORD_X, 0      ;COMPARA EL WREG AMB COORD_X, SI COORD_Y ES MES GRAN SALTA DE LINEA
1278        RETURN
1279
1280        MOVLW .0
1281        CPFSGT COORD_Y, 0      ;COMPARA EL WREG AMB COORD_X, SI COORD_Y ES MES GRAN SALTA DE LINEA
1282        RETURN
1283

```

Si l'atac es pot fer, convertim la coordenada en posició de la RAM amb la funció *CONVERT_COORD_TO_RAM*.

Després comprovem aquesta adreça del contrincant i si val 0 vol dir que hi ha aigua i anem a la funció *HAS_TOCAT_AIGUA_N*, però si val 1 vol dir que hi ha vaixell *HAS_TOCAT_BARCO_N*. Si no es cap d'aquests dos vol dir que ja està atacada aquesta posició i no fem res.

```

1284    ;CONVERTIR A RAM LA COORDENADA ACTUAL
1285    CALL CONVERT_COORD_TO_RAM
1286
1287    MOVFF RAM_COORD,FSROL      ;AGAFEM L'ADREÇA DE LA COORDENADA A ATACAR I LA CARREGuem AL PUNTER
1288
1289    ;COMPROVAR EL VALOR QUE TE 00 0 01
1290    MOVFF INDF0, VALOR ;COPIEM EL CONTINGUT DE LA RAM A LA VARIABLE CONTINGUT_RAM
1291
1292    MOVLW .0
1293    CPFSGT VALOR, 0      ;SI EL CONTINGUT DE LA RAM EL MAJOR A 00 S'HO SALTA
1294    GOTO HAS_TOCAT_AIGUA_N
1295    MOVLW .1
1296    CPFSGT VALOR, 0      ;SI EL CONTINGUT DE LA RAM EL MAJOR A 01 S'HO SALTA
1297    GOTO HAS_TOCAT_BARCO_N
1298    CLRF FSROL,0
1299    RETURN

```

En la funció *HAS_TOCAT_AIGUA_N* , l'únic que fem és carregar un 2 en aquesta adreça per indicar que és un atac fallat i cridar les funcions de guardar a l'EEPROM l'atac.

Finalment *HAS_TOCAT_BARCO_N* , carreguem un 3 a la posició de la RAM que toca, guardem l'atac a la EEPROM i després decrementem la vida del jugador enemic. Posem a 1 el 7e bit de **NUKE_TRYES** per indicar que hem tocat un vaixell i actualitzem les vides a l'EEPROM.

```

1300
1301    HAS_TOCAT_AIGUA_N
1302        ;POSAR LA POSICIO EN VERMELL
1303        MOVLW .2
1304        MOVWF INDF0, 0          ;POSEM 10 A LA COORDENADA ATACADA
1305        CALL GUARDAR_ATACK
1306        CALL GUARDAR_VIDES
1307
1308        RETURN
1309
1310
1311    HAS_TOCAT_BARCO_N
1312        ;POSAR LA POSICIO EN VERD
1313        MOVLW .3
1314        MOVWF INDF0, 0          ;POSEM 11 A LA COORDENADA ATACADA
1315        CALL GUARDAR_ATACK
1316
1317        BTFSC CURRENT_GRID ,0,0
1318        DECF PWM_S1,1,0
1319        BTFSS CURRENT_GRID ,0,0
1320        DECF PWM_SO,1,0
1321
1322        BSF NUKE_TRYS,7,0
1323
1324        CALL GUARDAR_VIDES
1325
1326        RETURN

```

```

Select an option:
1 - Global View
2 - Nuke
3 - Songs
> 2

> 2,2

[=] [X] [X] [ ] [ ] [ ] [ ] [ ]
[=] [X] [X] [ ] [ ] [ ] [ ] [ ]
[=] [X] [X] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ]

```

2.6 Opcional 6: El cançoner

En seleccionar l'opció del cançoner, anem a la funció SONG_OPTION que el que fà és primer mostrar el missatge de les notes que es poden introduir, tot seguit esperem que ens introduceixin les 5 notes. El que fem és carregar un 5 al comptador **COMPTADOR_LECTURA**, per fer el següent bucle de la lectura de les notes només 5 cops.

Inicialitzem el punter de la RAM a l'adreça 138 que és on començarem a guardar les 5 notes, mentre estem a l'espera que ens introduceixin les notes, també anem actualitzant la intensitat de la llum de la matriu led.

Dintre del bucle *d'ESPERA_SONG*, el que fem és esperar que ens arribi algun símbol pel RX comprovant el bit RCIF del registre PIR1, si val 0 tornem al bucle *ESPERA_SONG* i si val 1, mostrem el valor introduït per pantalla i llavors amb el **FSROL** l'hi carreguem 143 i l'hi restem el valor de **COMPTADOR_LECTURA**, d'aquesta manera apuntem a la regió de la RAM on volem guardar el valor.

Després l'hi restem 48 al valor introduït i el guardem, decrementem **COMPTADOR_LECTURA** i si val 0 vol dir que hem acabat d'introduir les 5 notes i sortim del bucle.

```

1329     SONG_OPTION
1330         CALL READ_FLASH_3
1331         MOVLW .5
1332         MOVWF COMPTADOR_LECTURA,0
1333         MOVLW .138
1334         MOVWF FSROL,0
1335     ESPERA_SONG
1336         MOVLW .5
1337         CPFSLT COMPT_LED,0
1338         CALL CONFIG_ADC_LED
1339
1340         BTFSS PIR1, RCIF,0
1341         GOTO ESPERA_SONG
1342         MOVFF RCREG,VALUE
1343         CALL MOSTRAR_CARACTER
1344         ;MOVFF CONTINGUT,VALUE
1345         MOVLW .143
1346         MOVWF FSROL,0
1347         MOVEF COMPTADOR_LECTURA,0
1348         SUBWF FSROL,1,0
1349
1350         MOVLW .48
1351         SUBWF VALUE,1,0
1352         MOVFF VALUE,POSTINCO
1353
1354         DECFSZ COMPTADOR_LECTURA,1,0
1355         GOTO ESPERA_SONG
1356

```

Finalment, fem un salt de linea, guardem la cançó a l'EEPROM i tornem a mostrar el menú d'opcionals.

```

1354      DECFSZ COMPTADOR_LECTURA,1,0
1355      GOTO ESPERA_SONG

1356
1357      MOVlw '\n'
1358      MOVWF VALUE,0
1359      CALL MOSTRAR_CARACTER
1360      MOVlw '\r'
1361      MOVWF VALUE,0
1362      CALL MOSTRAR_CARACTER
1363
1364      CALL GUARDAR_SONG
1365
1366      CALL READ_FLASH_2
1367
1368      RETURN
1369

```

A la flash és on tenim les 10 notes, el que guardem és el valor que carreguem a la variable **SOROLL_FREQ** que determina la freqüència del to cridant la funció SOROLL_ALTAVEU explicada a la fase 2.

Estan col·locats en ordre de to indicat, el primer valor 18 correspon a F4, el segon valor 37 a F3...

El que fem és iniciar el punter a TAULAVALORSNOTES i sumar-li l'ofset guardat a la RAM per seleccionar la nota introduïda.

```

TAULANOTES
    DB "\n\rNotes: F4 F3 E5 E4 C4 B3 A3 D4 A4 G4\n\r", "> ?"
TAULAVALORSNOTES
    DB .18,.37
    DB .9,.19
    DB .25,.26
    DB .29,.22
    DB .14,.16

```

Llavors quan cridem la funció *END_GAME* perquè s'ha guanyat una partida, inicialitzem el punter a 138 amb el **FSR2L**, llavors inicialitzem el punter de la flash a TAULAVALORSNOTES, l'hi sumem el primer valor de la RAM que pot ser de 0 a 9 dependent de la primera nota que s'haguí indicat, llegim el valor d'aquesta adreça a la flash amb *TBLRD**+ i el guardem a **SOROLL_FREQ** i cridem la funció **SOROLL_ALTAVEU** pel to durant 1 segon.

```

2014    END_GAME
2015
2016    ;CALL LECTURA_VALORS_EU_EP
2017
2018    SETF END_GAME_M,0
2019    MOVLW .138
2020    MOVWF FSR2L,0
2021
2022    CLRF VALOR_RAINBOW,0
2023    MOVLW b'00000001'
2024    MOVWF COMPT_RAINBOW,0
2025    MOVWF COMPT_RAINBOW_R,0
2026    CALL PINTAR_RAINBOW
2027
2028
2029    CALL INIT_FLASH_4
2030    MOVF POSTINC2,0
2031    ADDWF TBLPTRL,1,0
2032    TBLRD*+
2033    MOVF TABLAT,0
2034
2035    MOVWF SOROLL_FREQ,0
2036    CLRF COMPT_1S,0      ;FAREM SONAR LES 5 NOTES DE L'ALTAVEU UN COP ACABEM
2037
2038    CALL SOROLL_ALTAVEU ;1 NOTA
2039
2040    MOVLW .5
2041    CPFSLT COMPT_LED,0
2042    CALL CONFIG_ADC_LED

```

Ara el punter de la RAM estarà apuntant l'adreça 139 on tenim el valor de la segona nota i repetim el procés d'inicialitzar la flash, llegir el valor i fer el so. Això ho fem 5 cops en total per a fer els 5 tons.

```

2049
2050    CALL INIT_FLASH_4
2051    MOVF POSTINC2,0
2052    ADDWF TBLPTRL,1,0
2053    TBLRD*+
2054    MOVF TABLAT,0
2055
2056    MOVWF SOROLL_FREQ,0
2057    CALL SOROLL_ALTAVEU ;2 NOTA
2058

```

```

Select an option:
1 - Global View
2 - Nuke
3 - Songs
> 3

Notes: F4 F3 E5 E4 C4 B3 A3 D4 A4 G4
> 01236

```

3. Configuracions del microcontrolador

CONFIGURACIONS EUSART

Per poder transmetre i rebre dades de la PIC a l'ordinador, necessitem activar la comunicació EUSART.

Per aquest motiu hem configurat els registres TXSTA, RCSTA, BAUDCON, SPBREG i SPBREGH de la següent manera.

```
CONFIG_EUSART
    SETF TRISC, 0           ;inicialitzem tot el port C com a entrada

    BSF     TRISC, RC6, 0
    BSF     TRISC, RC7, 0

    BCF     TXSTA, TX9, 0
    BSF     TXSTA, TXEN, 0
    BCF     TXSTA, SYNC, 0
    BSF     TXSTA, BRGH, 0

    BSF     RCSTA, SPEN, 0
    BCF     RCSTA, RX9, 0

    BSF     BAUDCON, RCIDL, 0
    BCF     BAUDCON, RXDTP, 0
    BCF     BAUDCON, TXCKP, 0
    BSF     BAUDCON, BRG16, 0
    BCF     BAUDCON, WUE, 0
    BCF     BAUDCON, ABDEN, 0

    CLRF   BRGH, 0
    MOVLW  .87
    MOVWF  SPBRG, 0

    BSF     RCSTA, CREN, 0
    RETURN
```

Primer de tot hem declarat els pins RC6 i RC7 com a entrada, en aquests pins és on connectem el TX i RX.

Tot seguit, en el registre **TXSTA**, els bits més importants són, **TX9** que el posem a 0 per enviar dades de 8 bits, **TXEN** a 1 per habilitar el TX, el **SYNC** a 0 perquè volem una comunicació asíncrona, això ens permet enviar i rebre a la vegada, i finalment **BRGH** que depèn de la velocitat en la qual volem anar.

En el registre **RCSTA**, hem posat a 1 el bit **SPEN** que és l'enable de l'EUSART general, el RX9 ja per defecte està a 0 per rebre dades de 8 bits i **CPEN** a 1 per habilitar la recepció continua de dades

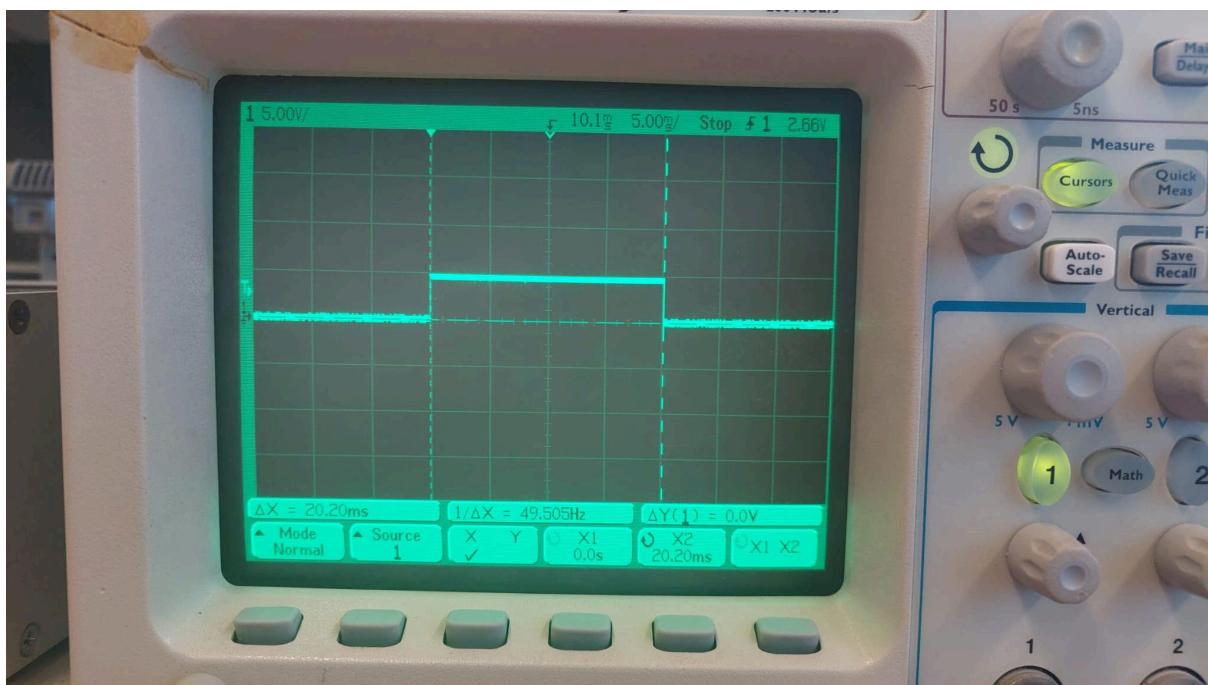
En el **BAUDCON** hi ha dos bits que per defecte ja estan a 0 que són RXDTP i TXCKP el que fan és negar la dada, no ho necessitem, finalment el BRG16 per establir la velocitat de comunicació.

Per al càlcul de la velocitat de comunicació, hem de complir un criteri que és que la matriu s'ha d'enviar tota entre 15 ms i 25 ms.

Primer hem mirat quan bits necessitem enviar per a mostrar una matriu, cada coordenada està formada per 3 símbols i hem d'enviar 64 coordenades i finalment cada símbol són 8 bits. Hem omès els salts de linea que fem cada 8 coordenades.

És a dir $64 \times 3 \times 8$ que és 1536, això és la quantitat de bits que hem d'enviar entre 15 i 25 ms, una cosa que no varem tindre en compte al fer el càlcul és que ho considerarem tot ideal, és a dir que no teníem en compte el temps que tardem a consultar la RAM, llegir el valor, etc.

Idealment, seria $1536/15\text{ms} = 102400$, $1536/25\text{ms} = 61440$. La velocitat que està entre aquests dos valors és 76800, en provar si complim amb els requisits en mostrar la matriu, tardàvem 26ms, per aquest motiu hem augmentat la velocitat a 115200 que en aquest cas sí que mostrem la matriu en 20ms aproximadament i estem dintre del marge.



Durant l'enviament de dades pel TX ens pot saltar la interrupció del timer 0 per a fer el PWM dels servomotors, però com que el màxim temps que hi estarem dintre de la interrupció són 2,5 ms. A aquest 20ms +- 2,5ms ja seguim dintre del rang així que no tenim problema.

Per a establir aquesta velocitat, apareix en la taula del datasheet i la que té l'error més baix equival a BRGH a 1, BRG16 a 1 i carregar un 87 en decimal al SPBRG.

CONFIGURACIONES ADC

Per a poder moure el cursor amb el Joystick, utilitzem l'ADC converter, també el fem servir per llegir el valor de la resistència regulable i controlar la intensitat de la llum dels leds de la matriu.

```
CONFIG_ADC
    MOVLW  b'00001100'
    MOVWF  ADCON1,0
    MOVLW  b'00101100'
    MOVWF  ADCON2,0
    RETURN
```

Primer hem configurat els registres ADCON1 i ADCON2, que només els configurem un cop al main i ja.

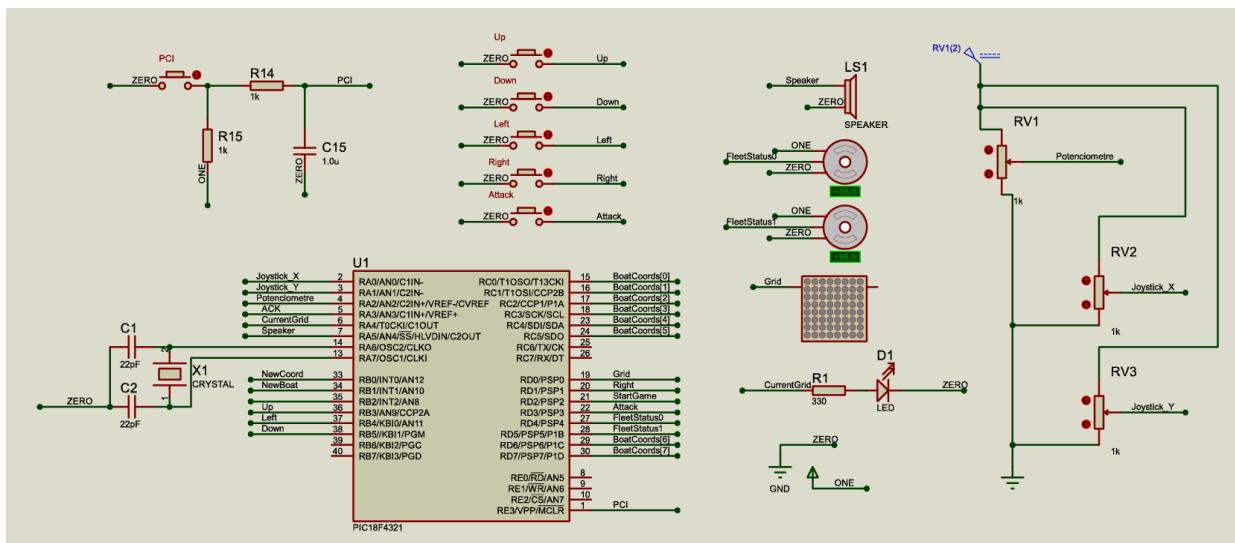
En l'ADCON1 en els bits VCFG1 i VCFG0 els deixem per defecte a 0 per indicar-li que les tensions de referència són les mateixes que les de l'alimentació.

Llavors en els bits PCFG[3..0] l'hi indiquem quants canals analògics volem utilitzar, per al Joystick necessitarem 2 i pel potenciómetre 1, així que l'hi carreguem 1101 per a indicar que el A0, A1 i A2 són analògics.

En el ADCON2 indiquem la velocitat de conversió, el TAD i la justificació de la conversió.

Hem seleccionat justificació a l'esquerra, 12 TADs i FOSC/4 perquè anem a una FOSC de 40MHz.

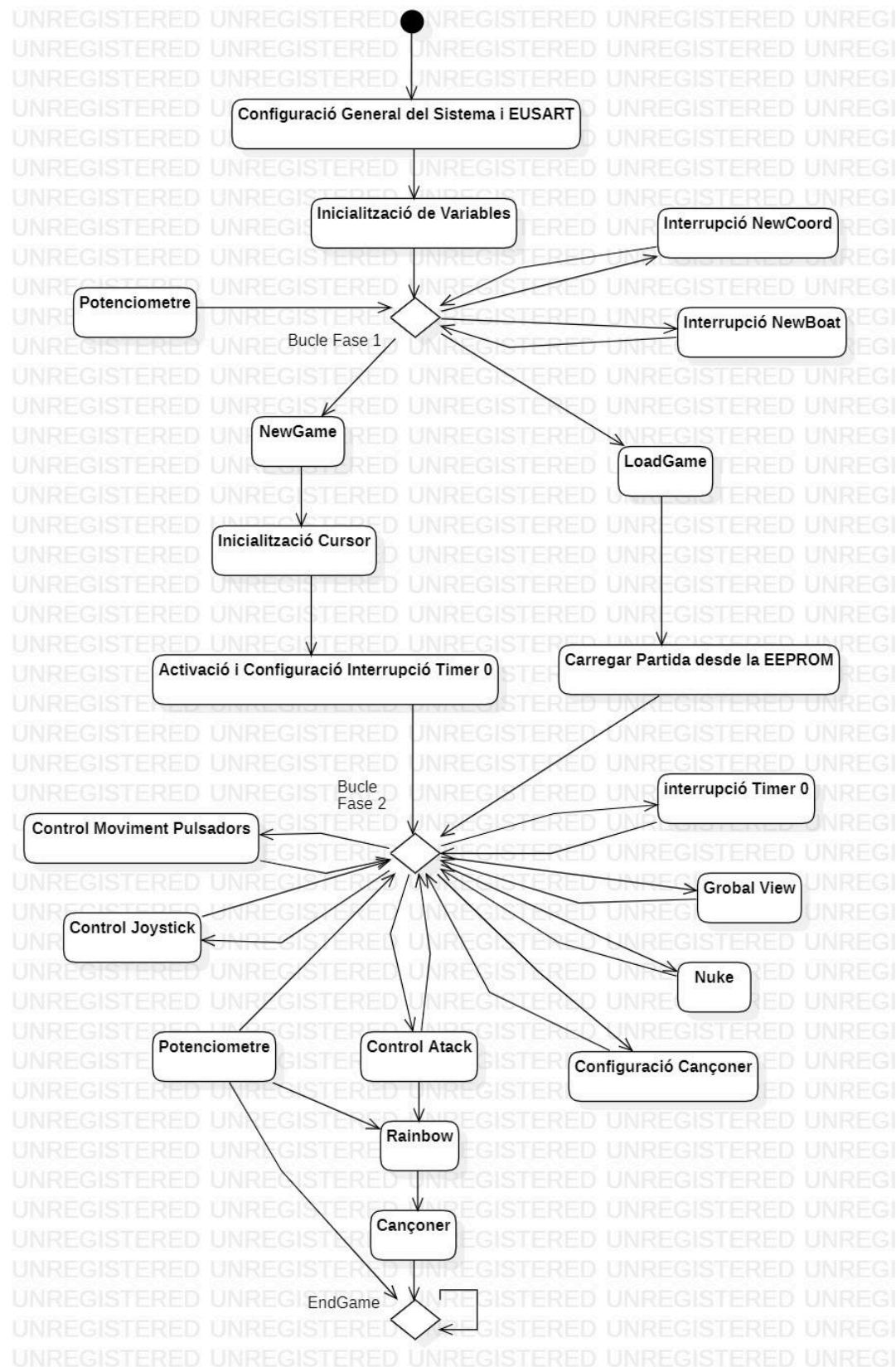
4. Esquema elèctric



Per a realitzar la fase 3, hem hagut d'afegir un joystick, que és representat per dos potenciómetres i també hem posat un potenciómetre per al control de la il·luminació de la matriu.

També hem reassiguat alguns ports del PIC per a poder utilitzar els canals analògics.

5. Diagrama d'activitat del software



6. Conclusions i problemes observats

Hem decidit fer tots els opcionals i el que més problemes ens ha donat és el de Brillantor de la pantalla variable, per a fer-ho no ens ha estat complicat, ja que només havíem de guardar el valor en una variable i anar-la llegint quan vulguem mostrar les matrius, però la part complicada ha sigut fer que es pugui modificar la brillantor en tot moment, durant la fase 1 en introduir els vaixells, fase d'atac, mentre fem les esperes, mentre fem els opcionals, en l'end game...

Hem hagut d'habilitar les interrupcions del timer 0 en la fase 1 per poder actualitzar la brillantor en temps real. El que no hem pogut aconseguir és fer que la brillantor s'actualitzi en temps real en fer els sons, perquè si anem actualitzant tan seguidament la matriu a la vegada que fem el so, aquest s'escolta distorsionat així que hem acabat no fent-lo en aquests moments.

A part d'això, ens ha迫çat a canviar la lògica en mostrar les matrius, nosaltres deshabilitavem les interrupcions cada cop que mostrem una matriu, però si volem que la brillantor sigui en temps real, anem mostrant la matriu cada 100 ms i deshabilitem massa cops les interrupcions, per aquest motiu en tocar els servos notàvem que els podríem moure una mica amb la mà.

El que hem fet és no desabilitar les interrupcions en cap moment canviant una mica la lògica, en l'únic moment ara on desabilitem les interrupcions és en l'escriptura de l'EEPROM.

També un altre problema que teníem és amb el joystick, ja que havíem de definir uns rangs de valors perquè es considerés que feia un moviment, i moltes vegades o bé no feia el moviment o bé es movia en diagonal. Per a solucionar-ho el que varem fer és ajustar els valors del rang i crear una variable flag, la qual al detectar un moviment o bé de l'eix X o bé de l'eix Y, s'activés, fent que no s'interpretessin més valors fins que s'executés el moviment.

En conclusió, hem estat capaços d'implementar les noves especificacions que demanaven a la fase 3, tant de la EUSART, com de la EEPROM i l'ADC. També, hem pogut implementar tots els extres de manera exitosa i sense problemes entre ells.

7. Planificació

Hores estimades i planificació:

| | Hores estimades | 05/02/24 | 12/02/24 | 19/02/24 | 26/02/24 | 4/03/24 |
|---------------------------|-----------------|----------|----------|----------|----------|---------|
| Comprensió de l'enunciat | 1 | | | | | |
| Disseny sobre paper | 5 | | | | | |
| Programació MPLAB | 15 | | | | | |
| Soldar | 1 | | | | | |
| Debugar | 8 | | | | | |
| Realització de la memòria | 4 | | | | | |

Hores totals dedicades:

| | Hores estimades | 05/02/24 | 12/02/24 | 19/02/24 | 26/02/24 | 4/03/24 |
|---------------------------|-----------------|----------|----------|----------|----------|---------|
| Comprensió de l'enunciat | 1 | | | | | |
| Disseny sobre paper | 3 | | | | | |
| Programació MPLAB | 18 | | | | | |
| Soldar | 2 | | | | | |
| Debugar | 4 | | | | | |
| Realització de la memòria | 5 | | | | | |

Per a la realització de la pràctica varem estimar unes hores en dissenyar i implementar la pràctica, però al final ha resultat ser una mica menys, ja que una vegada feta la part de la EUSART i la EEPROM, la implementació dels extres ha resultat ser més senzilla del que esperàvem, tot i això, varem haver de dedicar una setmana més a programar, ja que varem decidir fer tots els extres.

La part de soldadura, varen ser només el canvi de ports i el potenciòmetre a la placa, per tant, no va significar massa el temps de soldar.

Finalment, la part de debugar varem estar menys estona de l'esperada, ja que en tenir una bona distribució del codi, no es varen sorgir massa problemes, i la memòria ja l'hem realitzat en el temps esperat.