eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Culebrilla

Tiempo Real en Sistemas Empotrados

22 de enero de 2023

Daniel Cumbres, Asier Esteban y Unai Fernandez

Índice

1. Introducción

En este proyecto hemos replicado el mítico juego *"snake"*, utilizando una placa con un `STM32F411RET6` como microcontrolador. Para visualizar el juego se ha utilizado una matriz de LEDs de 8x8, y el control se realiza mediante las teclas del host.

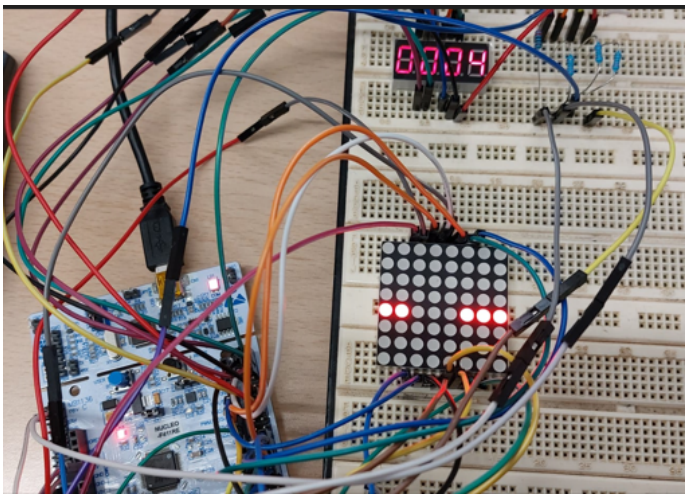

Figura 1: Circuito del proyecto culebrilla

2. Funcionalidades

Al encender la placa, se quedara esperando hasta que el usuario pulse el botón integrado en esta. Una vez se pulse el botón en la matriz de LEDs comenzaremos con una serpiente formada por un LED y un LED encendido que serán los puntos que deberemos alcanzar para aumentar el tamaño de nuestra culebra.

Para el control del juego el usuario deberá ejecutar la aplicación `SerialCom`, que le permitirá mover la serpiente pulsando las teclas A, S, D y W.

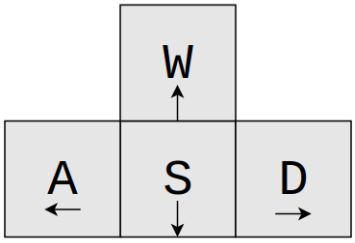

Figura 2: Teclas de control

La partida únicamente terminará cuando la culebra se muerda su propio cuerpo. Cada vez que la culebra consiga uno de los puntos, se incrementará el marcador en que se podrá ver mediante varios displays de 7 segmentos. Al final de la partida la puntuación final se quedará en el marcador hasta que el jugador inicie una nueva partida pulsando el botón integrado en la placa. Durante la partida tambien se puede pausar el juego pulsando este mismo botón.

3. Hardware

A continuación, pasamos a enumerar y describir la función de cada uno de los dispositivos utilizados:

3.1. Matriz de LEDs 8x8

En este elemento podremos visualizar nuestro juego de la serpiente.

Tal y como se muestra en la Figura 3, los pines positivos de los leds estan conectados a las filas y los pines negativos a las columnas. Es por eso que para encender un led se debe activar la fila correspondiente y desactivar las columnas en la que se quiera iluminar el led.

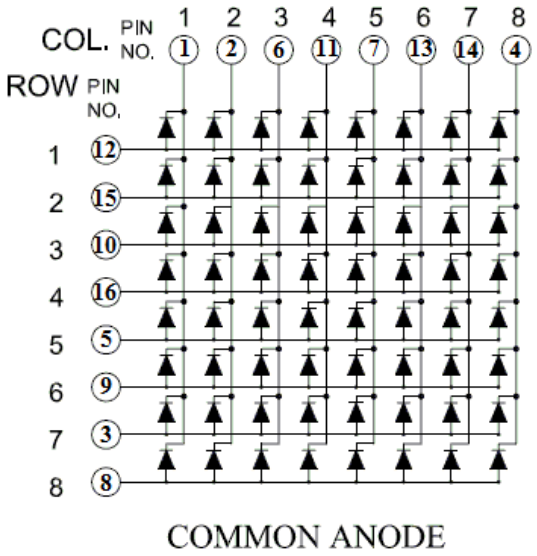

Figura 3: esquema de la matriz de 8x8 LEDs

Las conexiones que hemos establecido para el funcionamiento de la matriz son las siguientes:

Columnas	C0	C1	C2	C3	C4	C5	C6	C7
Pines Matriz	13	3	4	10	6	11	15	16
Salidas STM 32	PB9	PB8	PB7	PB5	PC1	PB6	PC0	PB0

Filas	F0	F1	F2	F3	F4	F5	F6	F7
Pines Matriz	9	14	8	12	1	7	2	5
Salidas STM 32	PB2	PB14	PB15	PB1	PB13	PC7	PA6	PA7

Figura 4: Pines asociados a la matriz de 8x8 LEDs

3.2. Display de 7 segmentos

En el display podremos visualizar diferentes mensajes una vez ejecutemos el proyecto:

- Hasta no pulsar el botón de inicio de partida se mostrará "- - - -" en el display.
- Durante el transcurso de la partida podrá ver la puntuación.

Para mostrar algo en el display se debe activar el pin del dígito correspondiente, así como los pines [A, B, C, D, E, F]. Sin embargo, como los dígitos están unidos, estos pines se mostrarán en todos los dígitos activos.

Las conexiones que hemos establecido para el funcionamiento del display son las siguientes:

Display	A	B	C	D	E	F	G	DP	DIG 1	DIG 2	DIG 3	DIG 4
Pines	11	7	4	2	1	10	5	3	12	9	8	6
Salidas STM 32	PC9	PC8	PC6	PC5	PA12	PA11	PB12	XX	PC10	PC12	PB5	PH1

Figura 5: Pines asociados a los displays

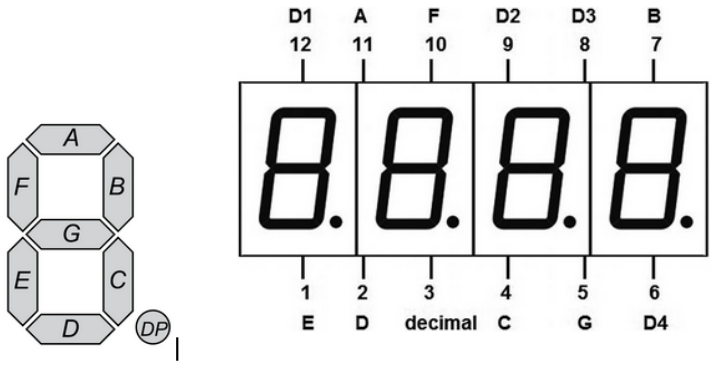

Figura 6

3.3. Pulsador Interno (Pausa/Reanudar/Iniciar partida)

Usaremos un único botón para comenzar la partida, pausar la partida y reanudarla. Usaremos la configuración de pull up para este botón. Las conexiones que hemos establecido para el funcionamiento del pulsador son las siguientes:

Pines	Positivo
Entradas STM 32	PC13

Figura 7: Pines asociados al pulsador interno

3.4. Esquema general

Tal y como se ve en la Figura 11 el host se comunica vía UART con la placa para controlar el movimiento y recibir información sobre la partida. Asimismo, en la placa se gestionan la lógica del juego y la puntuación que se mostrará en él display de 7 segmentos.

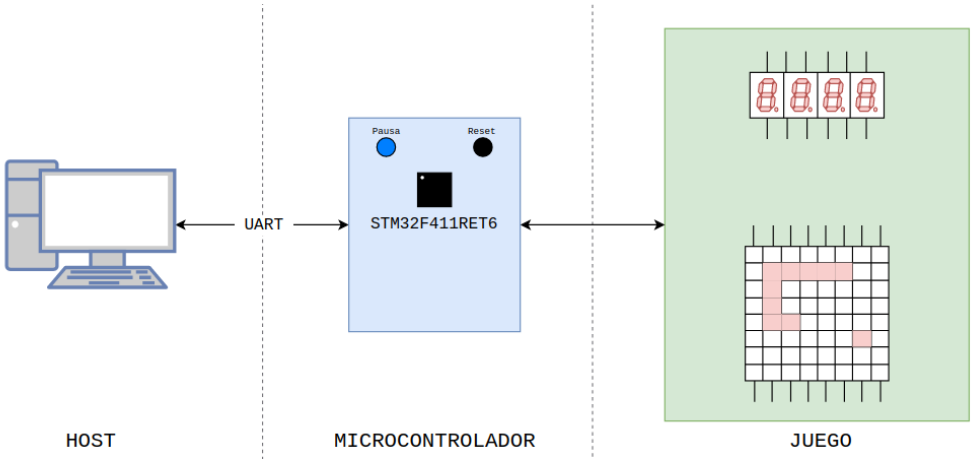

Figura 8: Esquema general del sistema

4. Descripción de las tareas

A continuación, enumeraremos y describiremos las tareas que hay en nuestro proyecto:

4.1. Juego

Esta será el núcleo del nuestro proyecto, en esta tarea se gestiona el juego de la culebrilla. Además de esto, esta tarea se comunica con el resto de tareas que explicaremos más adelante, para así poder mostrar en el transcurso de la partida el puntaje durante la partida (y al final de esta), la dirección que tiene que tomar la culebrilla enviada desde las teclas del teclado y el botón que permitirá arrancar, pausar y reanudar la partida. Hemos supuesto que esta tarea debe de tener una prioridad normal, ya que se estará ejecutando de manera continua hasta recibir una señal superior (como la del botón para parar la partida).

4.2. UART

Hemos creado dos tareas para poder llevar a cabo el protocolo de comunicación UART. Una de estas tareas (uart) la utilizamos para enviar información de la partida, como por ejemplo el puntaje obtenido en el transcurso de la partida y la otra tarea (uart rec) la usamos para recibir información. Concretamente la usamos para recibir que tecla del teclado se ha pulsado para poder cambiar el movimiento de la culebrilla.

Hemos decidido que debería de tener una prioridad alta, ya que ambas tareas tendrán que enviar y recibir información para poder proceder con el funcionamiento de la partida.

4.3. Botón

La tarea de botón será la tarea la encargada en dar comienzo, pausar y reanudar la partida. Por lo tanto, el juego de la culebrilla podrá iniciarse cuando este pulsador sea accionado y durante el transcurso de la partida si el botón ha sido pulsado, veremos como la culebrilla y el marcador se detienen en la última posición/valor que tenían respectivamente.

Hemos decidido que debería de tener una prioridad alta, como las tareas UART.

4.4. Board

Esta tarea es la encargada de mostrar en la matriz de leds la posición de la comida y la serpiente en tiempo real. Esta información está constantemente actualizada en la tarea de Juego 4.1 en una matriz del mismo tamaño que la de los leds, en donde se indica que leds tienen que estar encendidos y cuáles apagados.

Debido a las conexiones internas de la matriz (ver apartado 3.1), no se pueden activar dos leds en diagonal al mismo tiempo, ya que al activar dos filas y dos columnas se iluminan las 4 intersecciones de estas.

Es por eso que se ha implementado el método que se muestra a continuación, en el que se recorre la matriz y por cada fila se activan las columnas necesarias. de esta manera y con una tasa de refresco alta nos da la ilusión de ver múltiples leds encendidos al mismo tiempo.

Al final de la tarea se ha introducido un *HAL_Delay* de 5 milisegundos para evitar que la matriz de leds se sature e ilumine los leds contiguos.

```
for (i = 0; i < GRID; i++) {
    row_on(i, 1);
    //for (j=GRID-1; j>=0; j--) {
    for (j = 0; j < GRID; j++) {
        int b = board[i][j];
        if (b)
            col_on(j, 0);  //Encender
        else
            col_on(j, 1);  //Apagar
    }
    HAL_Delay(5);
    row_on(i, 0);
}
```

Hemos decidido que esta tarea debe de tener una prioridad normal, ya que se estará ejecutando de manera continua.

4.5. Display

La tarea display es la encargada en ir refrescando en todo momento el marcador que hayamos obtenido en el transcurso de la partida. Esta tarea está de manera paralela ejecutándose con la tarea de juego, ya que cada vez que la culebrilla come una comida este marcador debe de incrementar.

De la misma manera que en el apartado anterior, debido a las conexiones del display, no se pueden mostrar dos dígitos diferentes al mismo tiempo. es por eso que se usa el mismo método de activar un dígito y desactivarlo para mostrar el siguiente, generando la ilusión de tener todos los dígitos activos al mismo tiempo.

Hemos decidido que esta tarea debe de tener una prioridad normal, ya que se estará ejecutando de manera continua.

5. Comunicación entre las tareas

Mediante el siguiente esquema mostramos cómo será el envío y recepción de datos en las colas de mensajes en las tareas anteriormente descritas:

Figura 9: Comunicación entre las tareas

Como se puede observar en la figura anterior, la tarea del juego se comunica con la de transmisión de UART y con la del display. En esta cola se envía un `struct` que contiene todos los datos necesarios, la puntuación y la posición en el eje x e y de la cabeza de la culebrilla.

```
typedef struct valores {
    uint32_t eje[2];
    uint32_t joy[2];
    uint32_t puntos;
} valores_t;
```

5.1. Máquina de estados del proyecto

La máquina de estados que se ejecutara en nuestro proyecto seguirá los siguientes procesos:

Siendo P1 el botón que nos permite avanzar entre los distintos estados, hemos configurado el botón USER de la placa para que realice esta funcionalidad.

Durante la ejecución del juego, si la culebrilla adquiere la comida que aparece de manera aleatoria en la matriz y en el display superior aparece el puntaje de la comida que la culebrilla ha comido. El juego pasa al estado de pausado si se acciona nuevamente el pulsador P1, mostrando a su vez el puntaje de la partida obtenido hasta el momento. Si se vuelve a pulsar el botón P1, el juego conmuta al estado de ejecución nuevamente.

En caso de que durante la partida la culebrilla se muerda en una parte del cuerpo, la partida termina. Cuando esta finalice, se mantendrá por el display de 7 segmentos el puntaje obtenido en la última partida.

Para volver a comenzar otra nueva partida se debe de accionar nuevamente el pulsador.

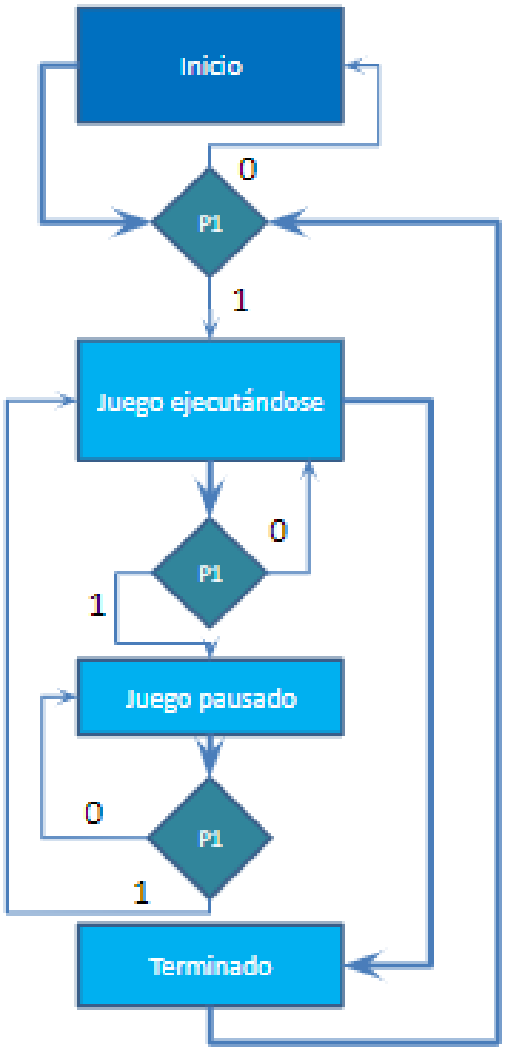

Figura 10: Máquina de estados

6. Implementacion del juego

Para esta aplicación, como se ha explicado antes, hemos desarrollado el juego del *snake*. La implementación de este juego se encuentra en los archivos de `culebrilla.c` y `culebrilla.h`.

Para la implementación de este juego se han definido 3 estructuras nuevas:

- **Coord:** es una estructura que define la coordenada x y la coordenada y.
- **Food:** es una estructura que contiene las coordenadas de la comida y una variable de control para saber si la han comido o no
- **Snake:** es una estructura que contiene un *array* de coordenadas de 8×8 y una variable de control para controlar la longitud de la serpiente

El juego comienza generando una serpiente y una comida en una posición aleatoria, y en una dirección preestablecida, en este caso, hacia la derecha. Mientras la serpiente siga viva se realizará el siguiente bucle.

Primero, se mueve la serpiente en la dirección que toque. Para ello, se utiliza la función *move_snake()* que tal y como se muestra en el código de abajo, calcula la siguiente posición del primer elemento de la longitud de la serpiente de manera circular, es decir, al llegar a un extremo aparecerá en el lado contrario.

```
coord_t head_next = snake->body[0];
    switch (dir) {
        case R:
            head_next.x = (head_next.x + 1) % GRID;
            break;
        case L:
            if (head_next.x == 0)
                head_next.x = GRID-1;
            else
                head_next.x--;
            break;
        case D:
            head_next.y = (head_next.y + 1) % GRID;
            break;
        case U:
            if (head_next.y == 0)
                head_next.y = GRID-1;
            else
                head_next.y--;
            break;
    }
```

Luego, se moverá el resto del cuerpo a la posición de la siguiente coordenada. De esta manera se genera la ilusión del movimiento de la serpiente. Además, antes de finalizar el movimiento se comprueba que la posición de la cabeza no coincida con ninguna coordenada de la serpiente.

```
for (i=snake->length-1; i>0; i--) {
    snake->body[i] = snake->body[i-1];
}
// check if is is dead
if (is_in_snake(&head_next, snake)) {
    snake->die = 1;
}
snake->body[0] = head_next;
```

Una vez que estén actualizadas todas las coordenadas de la serpiente, en la función *eat()*, se mira si la cabeza de la serpiente y la comida están en la misma posición. En este caso, se sumará 1 a la longitud de la serpiente y se indicará que se la comida ha sido consumida, lo que generará otra comida en una posición aleatoria que no esté ocupada por la serpiente.

```
if (snake->body[0].x == food->pos.x && snake->body[0].y == food->pos.y) {
    food->eaten = 1;
    snake->body[snake->length] = snake->body[0];
    snake->length ++;
}
```

Por último, la función *update_board()* guarda toda la información en una matriz de 8×8 que indica que posiciones se tienen que iluminar tal y como se explica en el apartado 4.4 Board de la descripción de tareas.

```
for (i=0; i<snake->length; i++) {
    s_pos = snake->body[i];
    leds_on(s_pos.x, s_pos.y);
}

leds_on(food->pos.x, food->pos.y);
```

7. SerialCom

`SerialCom` [6] es un programa multihilo, diseñado para comunicarse con la `STM32`, de forma bidireccional, por el puerto serie. La aplicación se va a encargar de leer los datos que se le envían y procesarlos, para mostrarlos en pantalla. A parte de eso, cada vez que se pulse una de las teclas de control, se enviará un comando a la placa, indicando la dirección a la que se tiene que mover la serpiente. Finalmente, se registrarán en un archivo todos los datos recibidos.

7.1. Componentes

La aplicación esta compuesta por tres hilos, uno para leer los datos que se mandan de la `STM32`, otro para enviar comandos a la `STM32` y por ultimo, un hilo para registrar todos los datos recibidos.

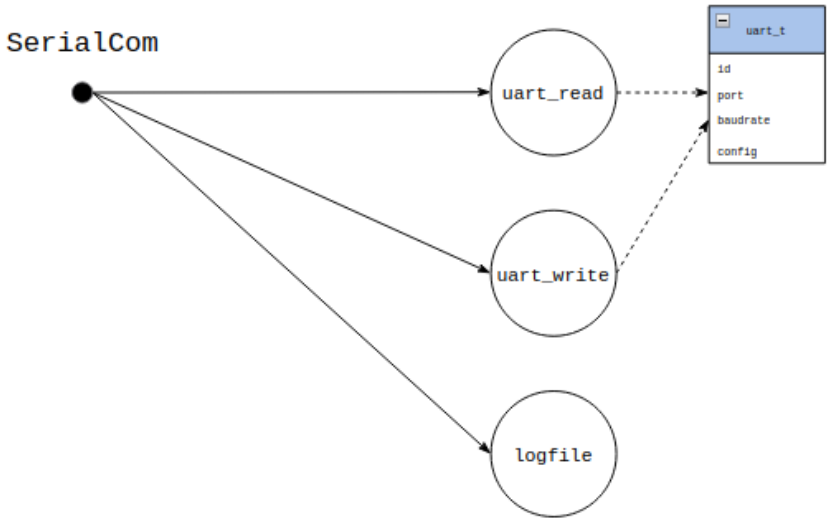

Figura 11: Esquema de los componentes de SerialCom

7.1.1. Programa principal

El programa principal se encargará de leer los parámetros, para configurar el programa de una manera u otra. La aplicación tiene dos parámetros, el *puerto* y el *baudrate*, la obtención de estos parámetros se realiza mediante el uso de `getopt`. Cada uno de los parámetros se podrá especificar con una versión larga y una corta.

Tras la obtención de parámetros se realiza la configuración del puerto serie. Para ello, se ha utilizado la API de Unix *termios*[1]. Para empezar con la comfiguración, se debe abrir el puerto serie con la llamada al sistema `open`.

```
    //Open /dev/ttyACM0
    if((uart.fd = open(uart.port, O_RDWR | O_NOCTTY | O_NDELAY)) < 0){
```

```
        perror("[!] ERROR: ");
        return -1;
    }
```

Después de abrir el puerto, este se configura utilizando las funciones o estructuras de datos que incluye *termios*. En nuestro caso para conseguir la configuración deseada simplemente se han asignedo diferentes valores a las variables dentro de la estructura de datos que ofrece *termios*.

```
    uart.config.c_cflag |= atoi(uart.baudrate); //baudrate
                                                //uart.config.c_cflag |= B115200;
    uart.config.c_iflag |= IGNPAR;      //no parity
    uart.config.c_cflag |= CS8;         //8 bit


    uart.config.c_cflag |= CREAD;       //Enable receiver
    uart.config.c_lflag &= ~(ICANON);       //Enable canonical mode

    uart.config.c_cc[VTIME] = 0;
    uart.config.c_cc[VMIN] = 1;
```

Finalmente para aplicar la configuración realizada, usamos el comando `tcsetattr()`, que ofrece *termios*.

```
    if(tcsetattr(uart.fd, TCSAFLUSH, &uart.config) < 0){
        perror("[!] ERROR (set attr): ");
        return -1;
    }
```

Una vez finalizada la configuración se crean los tres hilos necesarios para leer, escrebir y registrar de forma paralela.

7.1.2. uart_read

Como su nombre indica, este hilo se encarga de leer los datos que la placa le manda. Para la lectura del puerto serie se utiliza la llamada al sistema `read`. Una vez recibidos los datos se guardan en un buffer para procesarlos.

```
    if(uart_read(uart->fd, buff) > 0){
        //... (Procesar los datos recibidos)
    }
```

Los datos son 4 números separados por comas que indican, en este orden: la puntuación, la posición en el eje x, la posición en el eje y, y el modo de control. Los diferentes valores se separan y guardan en variables, para poder trabajar con ellos, usando la función `void parse_data(char * data)`.

Finalmente los datos procesados se meustran en pantalla utilizando la siguiente

plantilla.

```
------------------------- [GAME] -------------------------

    Points:                 x:                  y:

------------------------- [STATUS] -------------------------
- Port:
- Baudrate:
- Control:
```

7.1.3. uart_write

Este hilo sirve para mandar el comando correspondiente a la placa tras haber pulsado una de las teclas de control. Para obtener el valor de la tecla pulsada se ha utilizado la funcion `getch()` de la libreria *curses*. Esta función nos permiter leer el valor de la tecla sin tener que dar a enter después.

```
switch (getch()) {
        case 'a':
            sprintf(t, "ri");
            uart_write(uart->fd, t);
            break;
        case 's':
            sprintf(t, "do");
            uart_write(uart->fd, t);
            break;
        case 'd':
            sprintf(t, "le");
            uart_write(uart->fd, t);
            break;
        case 'w':
            sprintf(t, "up");
            uart_write(uart->fd, t);
            break;
        case 'o':
            endwin();
            exit(0);
            break;

    }
```

7.1.4. logfile

Este tercer hilo se encargar de registrar los datos que le llegan desde la `STM32` en un archivo llamado `log.txt`, usando llamadas al sistema como `open` y `write`.

7.2. Compilación y uso de la aplicación

Con la finalidad de facilitar este proceso el proyecto incluye un `Makefile` que se encargara de compilar y ejecutar `SerialCom` de una forma sencilla para el usuario.

Para compilar la aplicación, el usuario simplemente tiene que ejecutar el siguiente comando.

```
$ make
```

Para la ejecución, el usuario deberá introducir el siguiente comando, que ejecutara el programa para leer el puerto `/dev/ttyACM0` con una velocidad de 115200 *baud*.

```
$ make run
```

Si el usuario desea ejecutar la aplicación sin usar los parametros por defecto, debera hacerlo de la siguiente manera.

```
/bin/main -p /dev/ttyACM0 -b 115200
```

Las opciones `-p` y `-b` tienen una versión larga. Así seria el comando con dicha versión larga,

```
./bin/main --port /dev/ttyACM0 --baud 115200
```

8. Conclusiones

En general estamos contentos con el resultado obtenido, ya que hemos conseguido implementar la idea que habíamos planteado para el proyecto. A pesar de esto, el haber usado componentes que requerían tantos pines nos ha supuesto algún problema a la hora de utilizar estos pines como correspondían. Por otro lado, después de haber empezado a usar las colas de mensajes, los datos del juego no siempre se llegan a mandar bien del todo.

Referencias

[1] Serial programming/termios. Visitar web, 2022. (2022-12-25).

[2] Bugeados. Cómo hacer un reloj digital con arduino. Visitar web, 2020.

[3] S. H. O. CO. Data sheet, device number: Hl-m1588br. Visitar web, 2011.

[4] U. A. D. Culebrilla. https://github.com/UnaiFernandez/Culebrilla, 2022.

[5] U. Electronics. ¿cómo conectar y programar la matriz led 8×8 1088as? Visitar web, 2022.

[6] U. Fernandez. Serialcom. https://gitlab.com/unaiferr/serialcom, 2022.