Escuela técnica 32

Gral. Jose de san martin

Batalla

naval

Carpeta de campo del alumno:

Maximo Escobar

4-3

Computación

¿Qué es?

La batalla naval (juego de los barquitos o hundir la flota, nombre con el que se comercializó en España el juego de mesa; hundiendo barquitos, en algunos lugares de Hispanoamérica), del nombre en inglés "Battleship" , es un juego tradicional de estrategia y algo de suerte, que involucra a dos participantes.

Tableros

Los jugadores manejan un tablero de océano y un tablero de tiro; cada uno divididos en casillas.1​ Cada tablero representa una zona diferente del mar abierto: la propia y la contraria. En el primer tablero, el jugador coloca sus barcos y registra los «tiros» del oponente; en el otro, se registran los tiros propios contra el otro jugador, diferenciando los impactos y los que dan al agua. Al tiempo, se deduce la posición de los barcos del contrincante.

Naves

Al comenzar, cada jugador posiciona sus barcos en el primer tablero, de forma secreta, invisible al oponente.2​

Cada quien ocupa, según sus preferencias, una misma cantidad de casillas, horizontal y/o verticalmente, las que representan sus naves. Ambos participantes deben ubicar igual número de naves, por lo que es habitual, antes de comenzar, estipular de común acuerdo la cantidad y el tamaño de las naves que se posicionarán en el tablero. Así, por ejemplo, cinco casillas consecutivas conforman un portaaviones; cuatro, un buque; tres, un submarino; dos, un crucero; y una casilla aislada, una lancha. Los participantes podrían convenir, por ejemplo, colocar, cada uno, dos portaaviones, tres buques y cinco lanchas. No se podrán colocar barcos pegados entre sí en la misma dirección.

Desarrollo del juego

Una vez todas las naves han sido posicionadas, se inicia una serie de rondas. En cada ronda, cada jugador en su turno «dispara» hacia la flota de su oponente indicando una posición (las coordenadas de una casilla), la que registra en el segundo tablero. Si esa posición es ocupada por parte de un barco contrario, el oponente cantará ¡Averiado! (¡Toque!, ¡Tocado! o ¡Impacto!) si todavía quedan partes del barco (casillas) sin dañar, o ¡Hundido! si con ese disparo la nave ha quedado totalmente destruida (esto es, si la acertada es la última de las casillas que conforman la nave que quedaba por acertar). El jugador que ha tocado un barco en su anterior jugada, volverá a disparar hasta que falle. Si la posición indicada no corresponde a una parte de barco alguno, cantará ¡Agua!.

Cada jugador referenciará en ese segundo tablero, de diferente manera y a su conveniencia, los disparos que han caído sobre una nave oponente y los que han caído al mar: en la implementación del juego con lápiz y papel, pueden señalarse con una cruz los tiros errados y con un círculo los acertados a una nave, o con cuadrados huecos y rellenos, como se ve en la imagen. En la versión con pizarras, se utilizan pines de un color para los aciertos y de otro para las marras.

Fin de la batalla (fin del juego)

El juego puede terminar con un ganador o en empate:

Hay ganador: quien destruya primero todas las naves de su oponente será el vencedor (como en tantos otros juegos en los que se participa por turnos, en caso de que el participante que comenzó la partida hunda en su última jugada el último barco de su oponente que quedaba a flote, el otro participante tiene derecho a una última posibilidad para alcanzar el empate, a un último disparo que también le permita terminar de hundir la flota contraria, lo que supondría un empate);

Empate: si bien lo habitual es continuar el juego hasta que haya un ganador, el empate también puede alcanzarse si, tras haber disparado cada jugador una misma cantidad de tiros fija y predeterminada (como una variante permitida en el juego), ambos jugadores han acertado en igual número de casillas contrarias.

**Bibliografía**

**Códigos**

https://github.com/Dliang9/Arduino-Battleships

https://www.electrogeekshop.com/juego-de-batalla-naval-con-arduino-lcd-y-comunicacion-serial/

<https://github.com/calingeorgeadrian/Arduino-Battleships>

**Manejo**

https://www.youtube.com/watch?v=\_QNUIiuW2N0

<https://www.youtube.com/watch?v=okvUaG2BRBo>

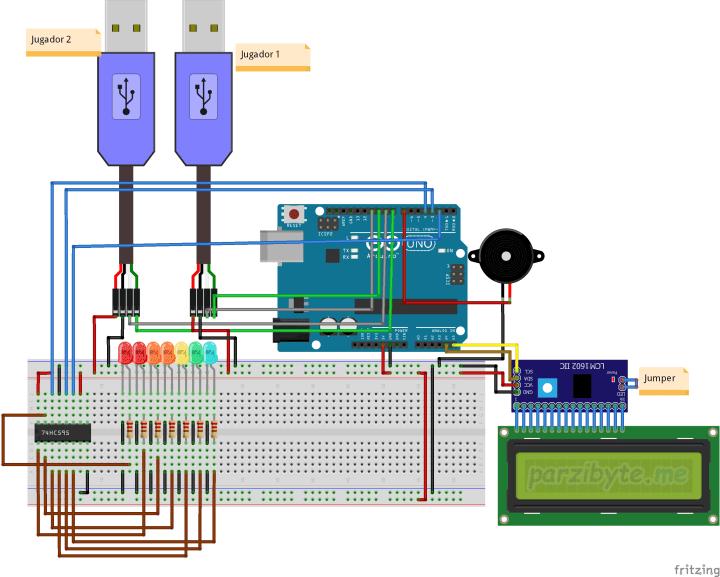
**Librerias mayormente utilizadas**

Wire.h

Liquidcrystal\_I2c

LedControl.h

Primeros días:

El proyecto tiene una gran cantidad de código lo q no permite una total libertad de manejo del mismo con el conocimiento aprendido, teniendo una base en fritzing el primer modelo digital estaba constituido de la siguiente manera

Utilizando leds y puentes para el correcto funcionamiento y el menor uso de cables para el trabajo, luego estaba pensado agregarle 2 display y 2 botones para señalar la posición tanto de los barcos como de los atacantes (el modelo de código era basado en un sistema de (X) y (Y) para un mejor entendimiento del mismo y por otras funciones agregadas), hay algunos sistemas de código agregados por cuenta propia que permitían hacer un plano de dibujo dentro del mismo LCD aunque se ocupaban 2 para el dibujo completo.

Durante las primeras 3 semanas el avance fue muy lento, ante la limitada selección de materiales a lo cual no permitía un avance total (ante los requerimientos mínimos del mismo proyecto)

Este sistema fue cambiado luego de recibir algunos componentes q serian prioritarios dentro del sistema, a los cuales serían el Joystick y pantalla matrix led 8x8 (2 unidades(solo una en funcionamiento)) a los cuales presentarían un manejo más atractivo del proyecto.

Funcionamiento de la matrix de leds:

El manejo de la misma dentro del código es una secuencia de bits que a través de vectores permiten mostrar una serie o un led prendidos e incluso mostrar figuras. Dentro del batalla naval el principal funcionamiento principal de la matrix, permite ver el punto exacto del mapeado a lo que consigue una mayor simplicidad en manejo de juego y un atractivo visual más llamativo.

Algún ejemplo de código de matrix de leds seria este:

byte ledStates[8] = {B01110000, B11111000, B11111100, B01111110, B01111110, B11111100, B11111000, B01110000};

Cada uno representa los leds y las columnas a prender

Avance luego de obtener los componentes necesarios.

Al obtener la matrix de leds, nos vimos obligados a cambiar el código sustancialmente y buscar uno nuevo mientras generábamos un modelo mejor programado y calificado, a lo que muchas opciones no eran eficientes, dado a que el modelo de código principal es muy pesado y está pensado para ser distribuido en varios Arduino uno o de otro tipo de Arduino como lo sería el Mega.

Anterior modelo utilizado

Durante los 2 meses acordados para hacer el proyecto, el modelo no ha mantenido muchos cambios, más que todo se basó en la obtención del conocimiento sobre los componentes nuevos ya que no teníamos información concreta de su uso. Por ende, el comienzo de la segunda parte del proyecto se vería en la segunda semana de noviembre presentando el primer uso verídico del programa en sí.

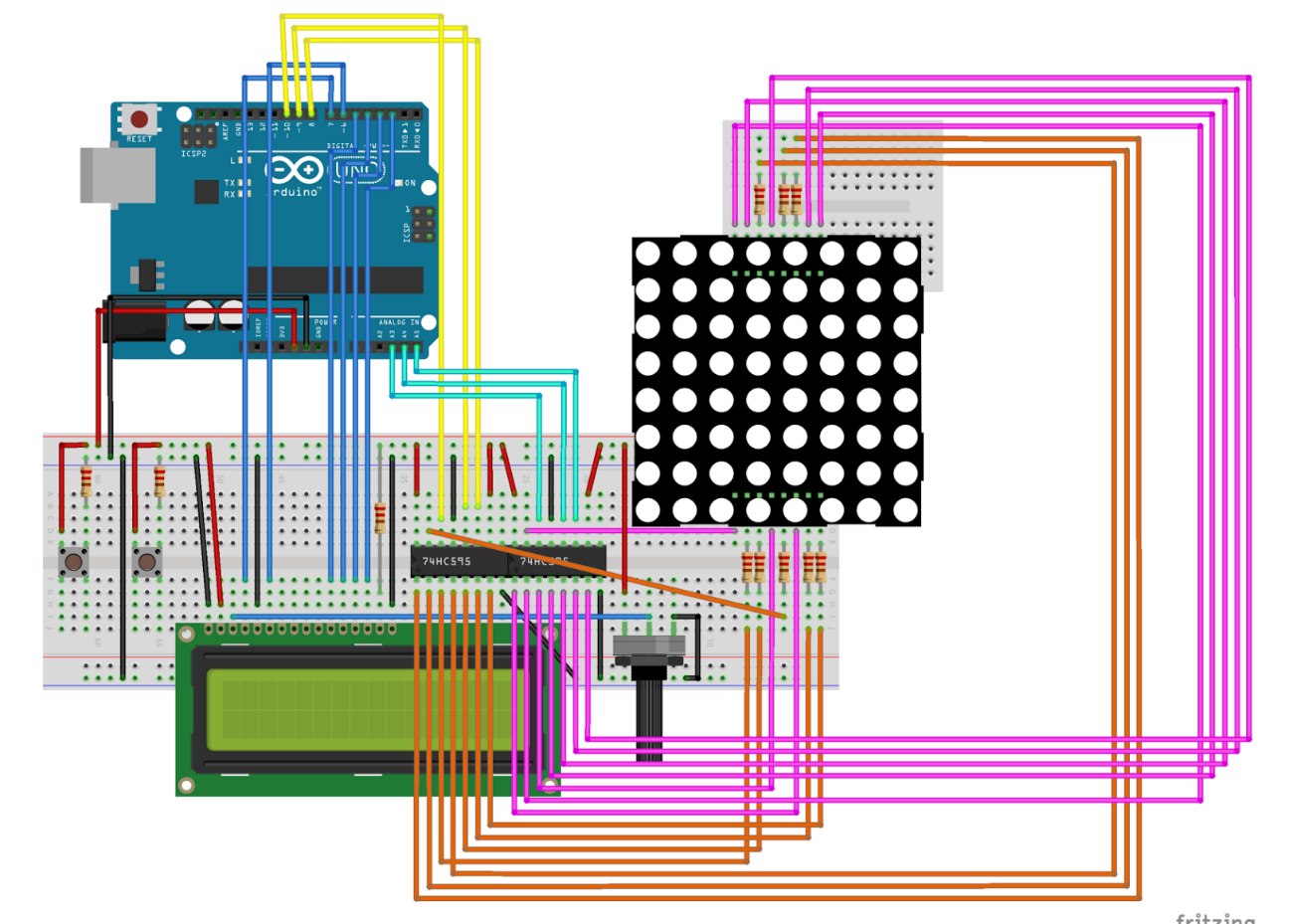
El primer comienzo del uso fue la programación de la pantalla LCD pero ahora utilizando un puente que permite menos conexiones al Arduino (utilizando la librería Liquidcrystal-I2C) . Y de la misma manera un puente para la matriz de leds q ejecutaba el mismo proceso q el puente de la LCD, con la diferencia en que uno estaba conectado a 2 pines analógicos y el otro a 3 pines digitales.

El principal cambio pensado era sustituir los componentes de botones por una touch pad (cosa que al final no se conseguirá lograr). Aunque el modelo mantenía la simpleza de 2 botones 2 matrix de leds y una lcd.

Debido al peso del programa y varios errores dentro del mismo, este se tuvo q descartar, pero una pequeña muestra del código seria esta:

#include <Wire.h>  
#include <LiquidCrystal\_I2C.h>  
LiquidCrystal\_I2C lcd(0x27,20,4);  
  
int redButtonPin = 12;  
int yellowButtonPin = 13;  
int playerBoard[8][8];  
  
int colClockPin = 9;   
int colLatchPin = 11;  
int colDataPin = 10;    
  
int rowClockPin = 6;  
int rowLatchPin = 7;  
int rowDataPin = 8;  
  
byte ledStates[8] = {B01110000, B11111000, B11111100, B01111110, B01111110, B11111100, B11111000, B01110000};  
byte GroundLEDs [8] = {B01111111, B10111111, B11011111, B11101111, B11110111, B11111011, B11111101, B11111110};   
  
int enemyBoard[8][8];  
int spotsBombed[8][8];  
  
int hitCounter = 0;  
int placesToHit = 0;  
  
void setup() {  
  pinMode(redButtonPin, INPUT);  
  pinMode(redButtonPin, INPUT);  
  lcd.init();     
  lcd.backlight();  
  lcd.setCursor(0,0);  
  pinMode(rowLatchPin, OUTPUT);  
  pinMode(rowClockPin, OUTPUT);  
  pinMode(rowDataPin, OUTPUT);  
    
  pinMode(colLatchPin, OUTPUT);  
  pinMode(colClockPin, OUTPUT);  
  pinMode(colDataPin, OUTPUT);        
    
  digitalWrite(colLatchPin, LOW);  
  shiftOut(colDataPin, colClockPin, MSBFIRST, B11111111);    
  digitalWrite(colLatchPin, HIGH);  
  
  for (int i = 0; i < 8; i++) {  
    for (int j = 0; j < 8; j ++) {  
      playerBoard[i][j] = 0;  
      spotsBombed[i][j] = 0;  
      enemyBoard[i][j] = 0;  
    }  
  }  
  placesToHit = 9;  
  
  Serial.begin(9600);  
  printStringScrolling("Soy don dimadon");  
  printStringScrolling("Dueño del domodin de dimadon ");  
  printStringScrolling("Heredero de la fortuna");  
  printStringScrolling("Del domodim dinsdale");  
  setBoat(2);  
  printStringScrolling("Los barquitos existen :D");  
  printStringScrolling("The Game");  
  setBoat(3);  
  printStringScrolling("WUUUUUUUUUUJUUUUUUUU ");  
  printStringScrolling("ñam");  
  setBoat(4);  
  
  communicateBoards();  
  printStringScrolling("Finished setting up board, now you can play, may the best bomber win ");  
}

Imagen del modelo en fritzing (1 jugador)



Una pequeña constitución del serial print y algunos de los pines a utilizar dentro del programa de la batalla naval (obviamente el código era mucho más extenso de un aproximado de 625 líneas de código).

Modelo actual trabajado

En si el modelo no paso por cambio a nivel estético, 1 Arduino mega o 2 Arduino uno para una buena ejecución (debido al mismo peso del código), pantallas lcd para mostrar algunos de los datos y las matrix de leds… Aunque en este momento se incluye un sistema de códigos y funciones separadas para un mejor control y mejor entendimiento de la complejidad del código y utilizar los dos joysticks (con fallas en medio de algunos procesos). Se ajusta bastante los métodos de utilización de la matrix 8x8 mostrando mas individualmente los contenidos del mismo en torno al juego y un apartado más intuitivo y más móvil que antes e incluso con interacciones mas atractivas utilizando un buzzer q tiende a llamar más la atención.

Dentro de algunos apartados dentro del código podemos encontrar:

// Ganador

byte youWon[8][8] = {{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 1, 0, 0, 0, 0, 1, 0},

{0, 0, 1, 1, 1, 1, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0}

};

// Perdedor

byte youLost[8][8] = {{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0},

{0, 0, 1, 0, 0, 1, 0, 0},

{0, 0, 0, 0, 0, 0, 0, 0},

{0, 0, 1, 1, 1, 1, 0, 0},

{0, 1, 0, 0, 0, 0, 1, 0},

{0, 0, 0, 0, 0, 0, 0, 0}

};

O en su defecto

player1.shipsNumber = 4;

player1.shipsLeftToPlace = 4;

player1.isReady = false;

player1.pressedStart = false;

player1.wantsToRestart = false;

player1.X\_PIN = X\_PIN1;

player1.Y\_PIN = Y\_PIN1;

player1.PLACE\_PIN = PLACING\_PIN1;

player1.ROTATE\_PIN = ROTATE\_PIN1;

player1.oldX = 0;

player1.oldY = 0;

player1.newX = 0;

player1.newY = 0;

player1.placeBtnState = LOW;

player1.lastPlaceBtnState = LOW;

player1.rotateBtnState = LOW;

player1.lastRotateBtnState = LOW;

player1.targetsLeft = 4 + 3 + 2 + 2; // suma longitude target barcos

player1.ships = new ship[4];

player2.shipsNumber = 4;

player2.shipsLeftToPlace = 4;

player2.isReady = false;

player2.pressedStart = false;

player2.wantsToRestart = false;

player2.X\_PIN = X\_PIN2;

player2.Y\_PIN = Y\_PIN2;

player2.PLACE\_PIN = PLACING\_PIN2;

player2.ROTATE\_PIN = ROTATE\_PIN2;

player2.oldX = 0;

player2.oldY = 0;

player2.newX = 0;

player2.newY = 0;

player2.placeBtnState = LOW;

player2.lastPlaceBtnState = LOW;

player2.rotateBtnState = LOW;

player2.lastRotateBtnState = LOW;

player2.targetsLeft = 4 + 3 + 2 + 2; // Suma de longitudes de los barcos

player2.ships = new ship[4];

Cabe destacar que aun faltan monitorear algunas funciones (ya que el código principalmente estaba en otro idioma y no permitía entender bien, por ende, se tuvo q traducir en su mayoría casi todo)

Este modelo no cuenta con imágenes por dos razones, es muy reciente, el archivo original no presenta imágenes y tuve ciertos problemas con el pc que me impidieron retomar muchos archivos previamente guardados.